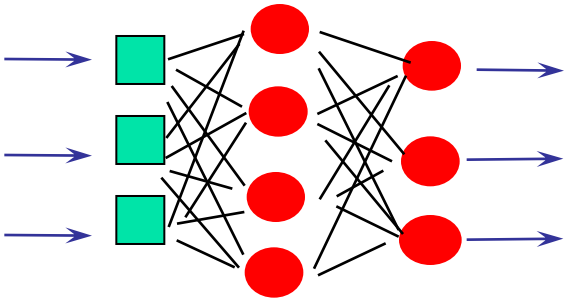
Redes Neurais



Prof. **Eduardo Raul Hruschka**

(Slides adaptados dos originais elaborados pelos Professores André C. P. L. F. de Carvalho e Rodrigo F. Mello)

Principais tópicos

- Por que estudar Redes Neurais (RNs)?
- O que são RNs?
- Como funcionam?
- Principais modelos

Redes Neurais (RNs)

- Inspiração biológica: animais são capazes de se adaptar às mudanças em seus ambientes, usando seus sistemas nervosos;
- Sistemas nervosos: formados por unidades simples (neurônios).
- Simulação das suas funções, por meio de modelos, pode produzir respostas similares em *sistemas artificiais*.
  - → Redes Neurais / Redes Neuronais Artificiais.
- Cérebro humano: ≅10 bilhões de neurônios (cada um conectado em média a outros 10^4) / eventos neuronais acontecem em 10^{-3} segundos;
- Eventos em circuitos de silício acontecem na ordem de 10^{-9} segundos;
- Dois grandes *grupos de pesquisa*:
  - Usar ANNs para estudar processos de aprendizagem biológica;
  - Desenvolver algoritmos de aprendizado automático.

Alguns marcos históricos:

- McCulloch & Pitts (1943) – rede formada por neurônios modelados pela "lei do tudo ou nada" poderia computar *qualquer função computável;*
- Hebb (1949) – formulação explícita de uma regra de aprendizagem fisiológica para modificação sináptica;
- Minsky (1954) – estendeu os resultados de McCulloch & Pitts;
- Rosemblatt (1958) – perceptron;
- Minsky & Papert (1969) – limitações dos perceptrons de uma única camada. Sugeriram que não haveria motivo pra acreditar que perceptrons de múltiplas camadas poderiam superar suas limitações....

  $\rightarrow$ Diminuíram significativamente os trabalhos sobre RNs.
- Bryson & Ho (1969) / Werbos (1974) / Parker (1985) / LeCun (1985) / Rumelhart, Hilton, Williams (1986) ;
- Hopfield (1982) – "física com redes neurais";
- Kohonen (1982) - mapas auto-organizáveis ;

Por que Redes Neurais?

- Trabalhos em RNs começaram com desejo de entender o cérebro (e copiar seu funcionamento).

- Sucessos iniciais foram encobertos
  - Promessas exageradas sobre a capacidade dos primeiros modelos

O que são Redes Neurais

- RNs são modelos de computação com propriedades particulares
  - Adaptar ou aprender
  - Generalizar
  - Agrupar ou organizar dados

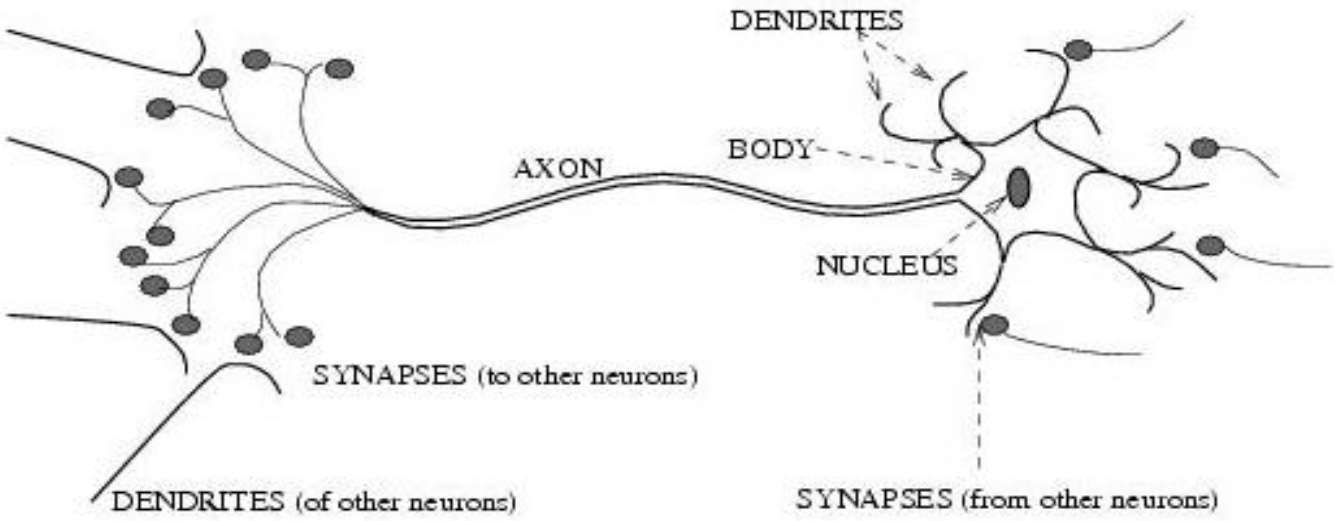
O que são Redes Neurais

- Modelos computacionais distribuídos inspirados no cérebro humano
  - Compostas por várias unidades de processamento ("neurônios")
  - Interligadas por um grande número de conexões ("sinapses")

Características das RNs

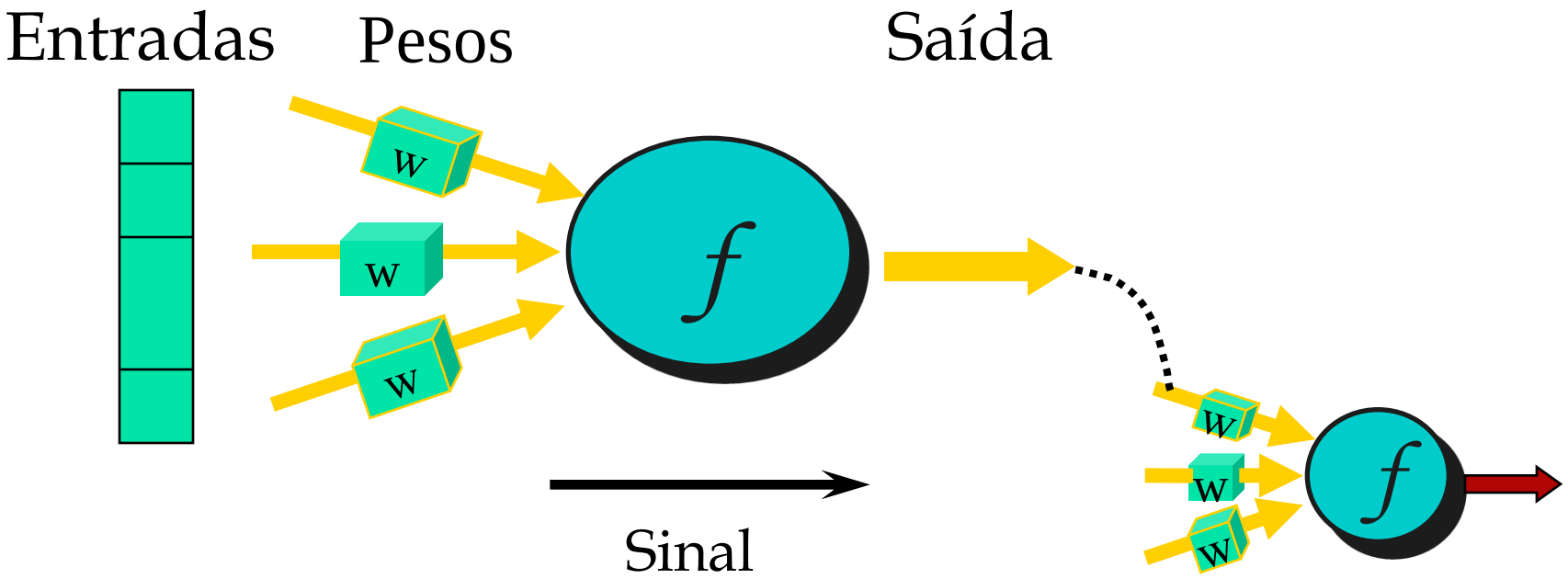
- Aprendem por meio de exemplos
  - Inferência estatística não paramétrica
- Adaptabilidade
  - Dilema plasticidade X estabilidade
- Capacidade de generalização
  - Dilema polarização X variância
- Certa tolerância a falhas

Neurônios:

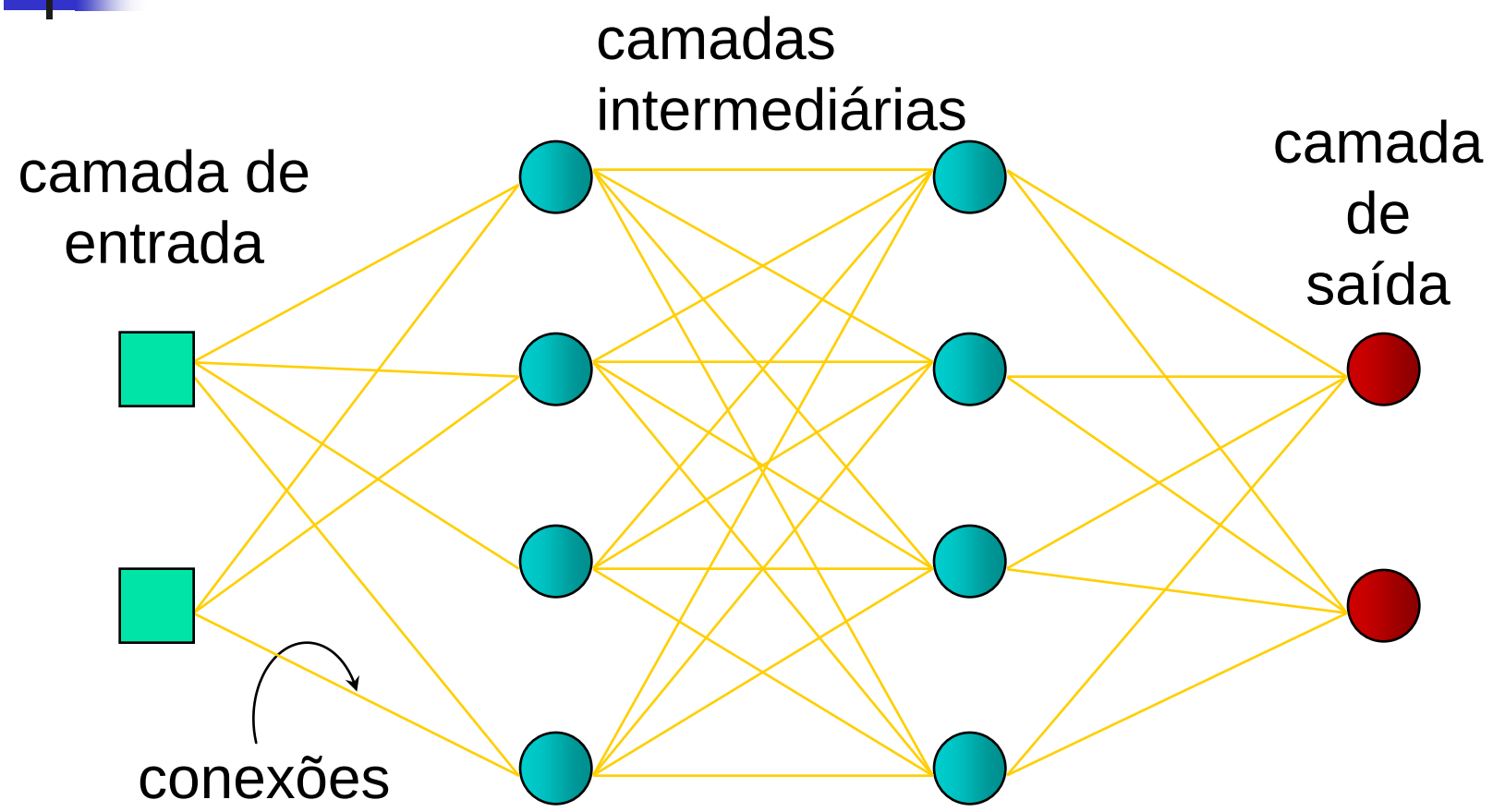


- **Dendritos:** recebem impulsos nervosos (informação) de outros neurônios e conduzem-nos ao corpo da célula;
- **Corpo da Célula**: informação é processada e novos impulsos são gerados;
- **Axônio**: transmitem os impulsos gerados no corpo da célula a outros neurônios.
- **Sinapse:** ponto de contato entre as terminações dos axônios e os dendritos. Controla a transmissão de impulsos, proporcionando a capacidade de adaptação do neurônio.

Modelo de neurônio artificial



Estrutura mais geral



Conceitos básicos

- Principais aspectos das RNs
  - Arquitetura
    - Unidades de processamento (nodos)
    - Conexões
    - Topologia
  - Aprendizado
    - Algoritmos
    - Paradigmas

Unidades de processamento

- Neurônios, nós, nodos, unidades
- Recebem entradas de unidades A;
- Computam uma função sobre entradas e enviam o resultado para unidades B.

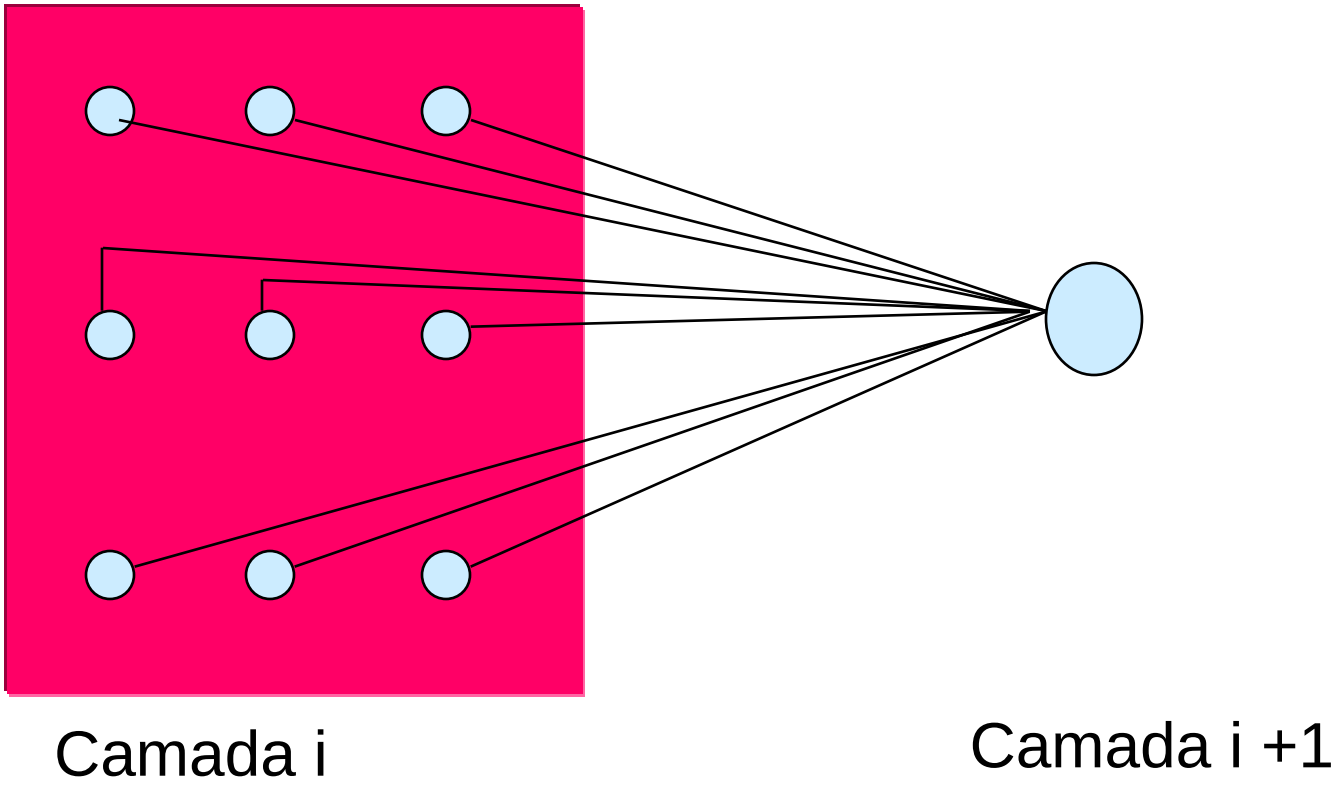
Conexões

- Forma de interligação entre neurônios
- Codificam o que a RN aprendeu
- Tipos de conexões em função dos seus pesos (w)
  - Excitatória: $w > 0$
  - Inibitória: $w < 0$

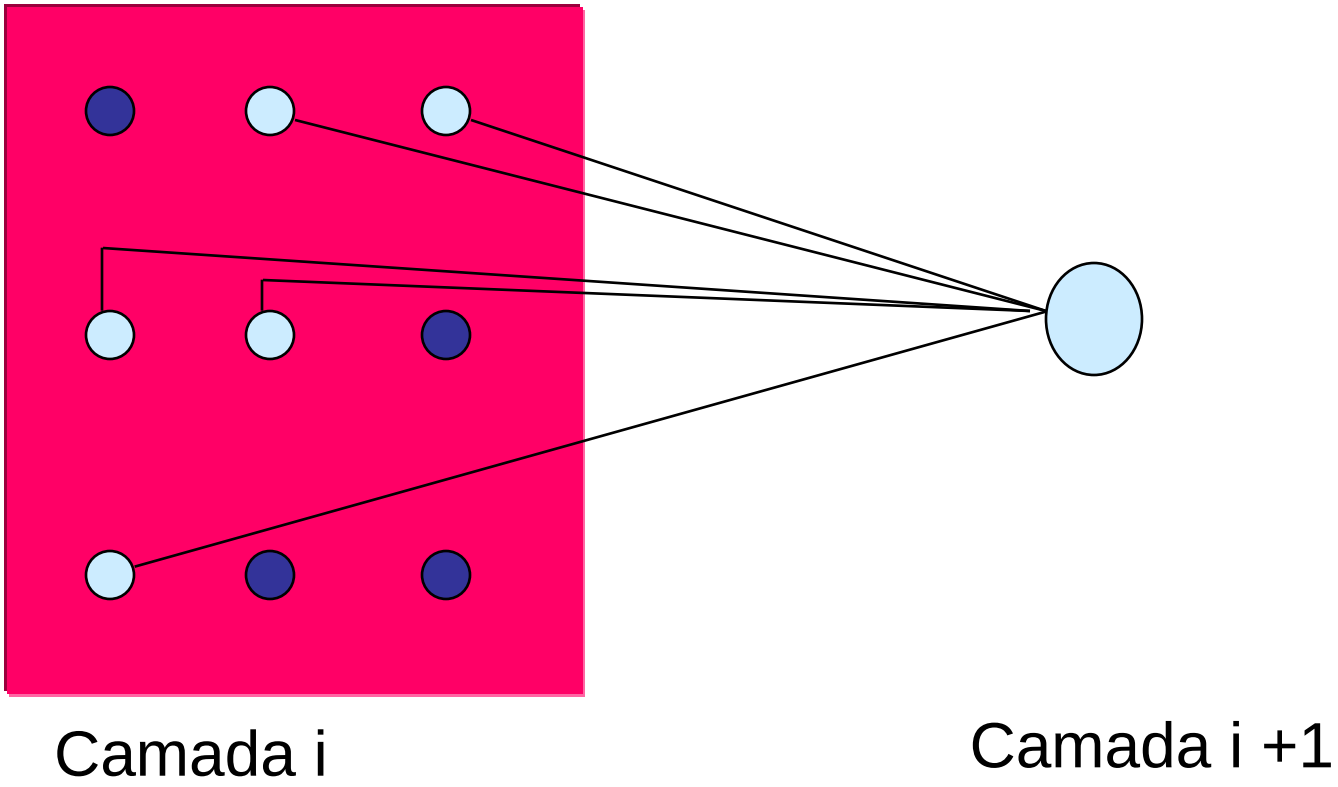
Topologia

- Número de camadas
  - Uma camada (e.g., Perceptron, Adaline)
  - Multi-camadas (e.g., MLP, RBF)
    - Completamente conectada
    - Parcialmente conectada
    - Localmente conectada

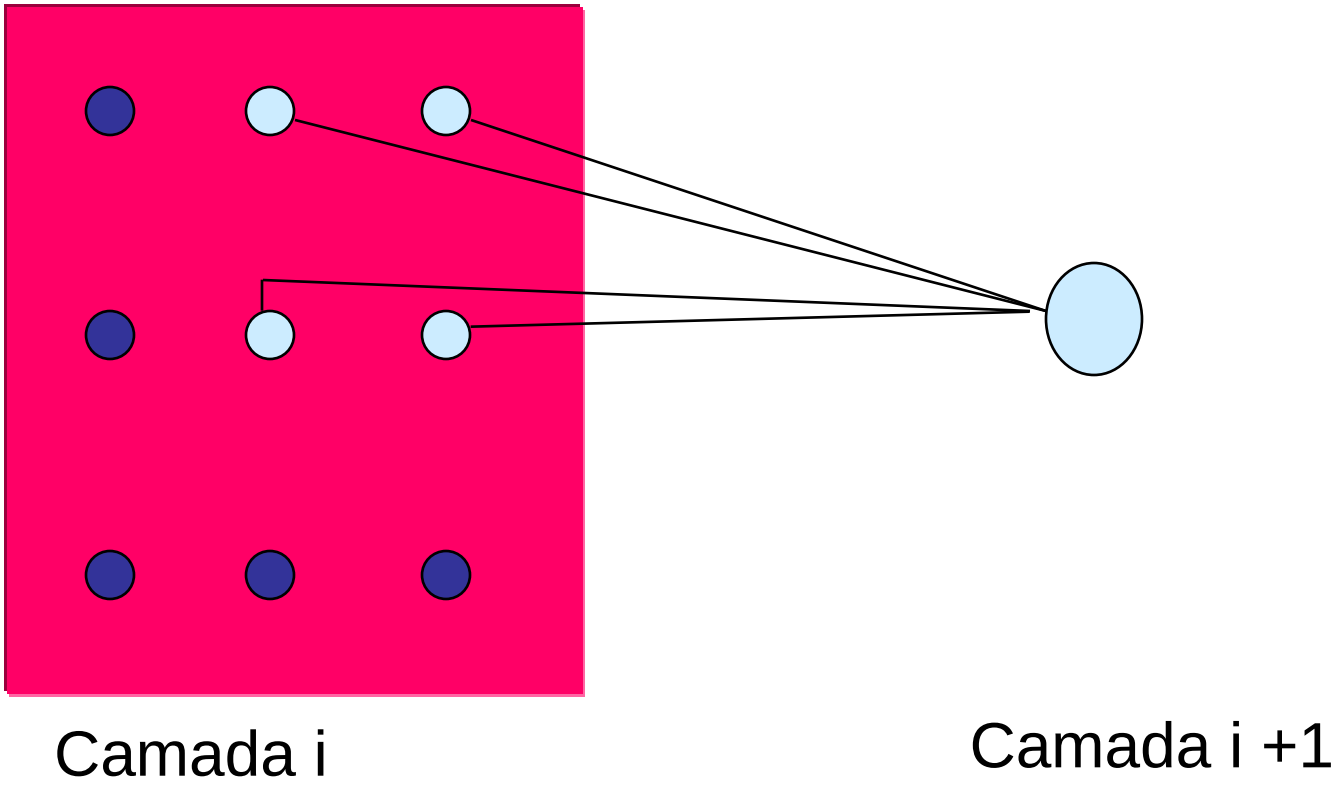
Completamente conectada



Parcialmente conectada



Localmente conectada

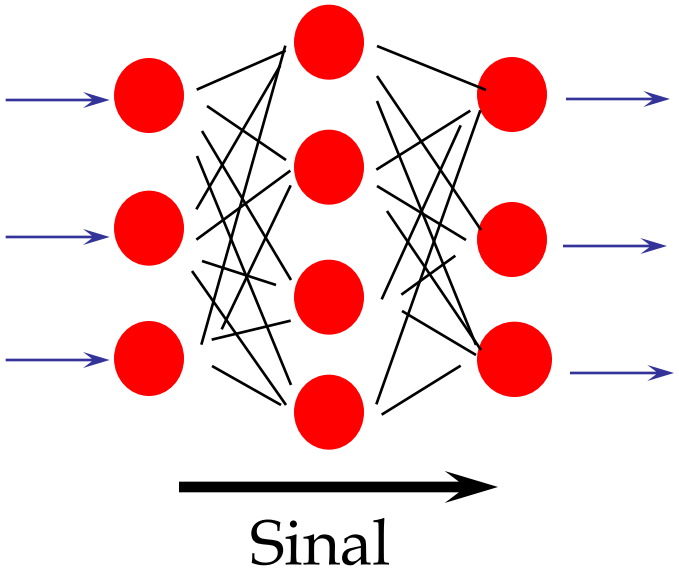


Topologia

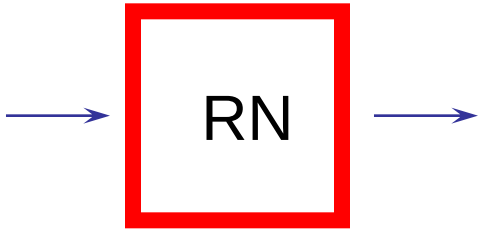
- Arranjo das conexões
  - Redes *feedforward*
    - Não existem *loops* de conexões
  - Redes recorrentes
    - Conexões apresentam *loops*
    - Mais utilizadas em sistemas dinâmicos
  - Grades
    - Matriz n-dimensional de nodos

Redes feedforward

- Sinais seguem em uma única direção

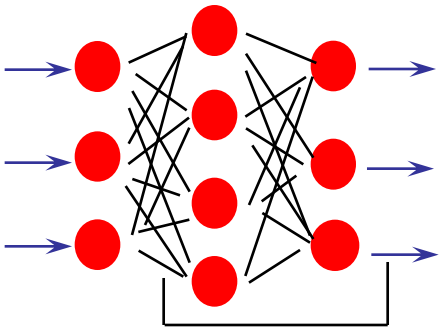


OU

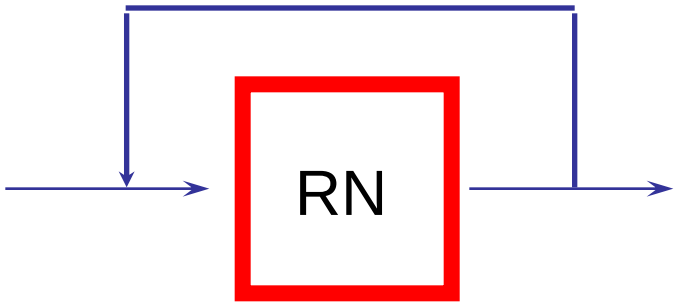


Redes recorrentes

- Conexões ligando a saída da rede à sua entrada



OU



- Podem se *lembrar* de entradas passadas e, consequentemente, processar informações sequenciais (no tempo ou espaço)

Aprendizado

- Algoritmos de aprendizado
  - Conjunto de regras bem definidas para a RN aprender a resolver um dado problema
  - Podem ser agrupados em quatro categorias
    - Correção de erro
    - Hebbiano
    - Competitivo
    - Termodinâmico (Boltzmann)

Aprendizado

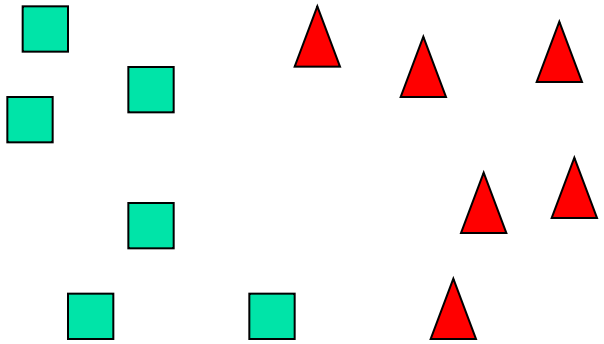
- Paradigmas de aprendizado
  - Supervisionado
  - Não supervisionado
  - Reforço
  - Híbrido

Perceptron

- Desenvolvida por Rosemblat, 1958
- Neurônio de McCulloch-Pitts (1943)
- Rede mais simples para classificação de padrões linearmente separáveis

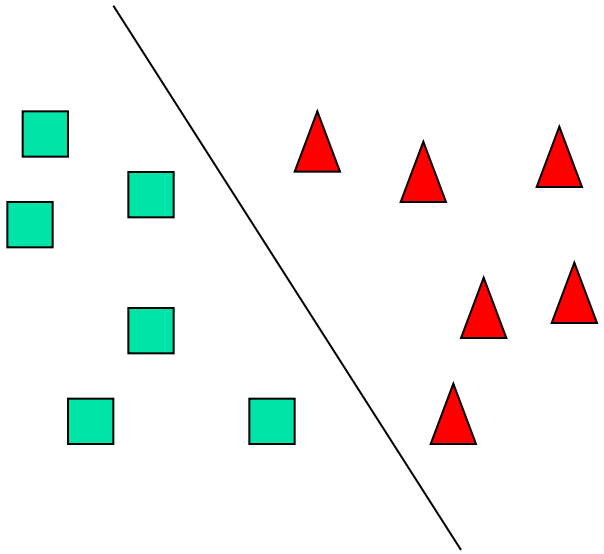
Perceptron

- Padrões linearmente separáveis

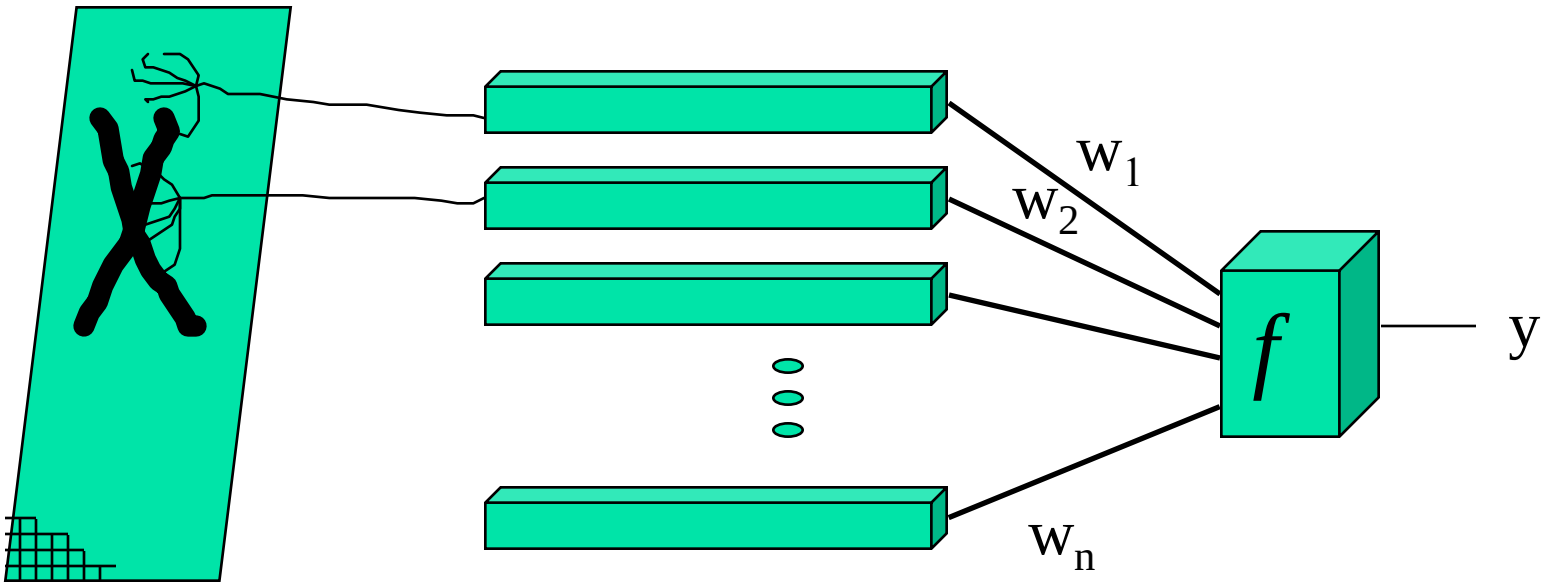


Perceptron

- Padrões linearmente separáveis

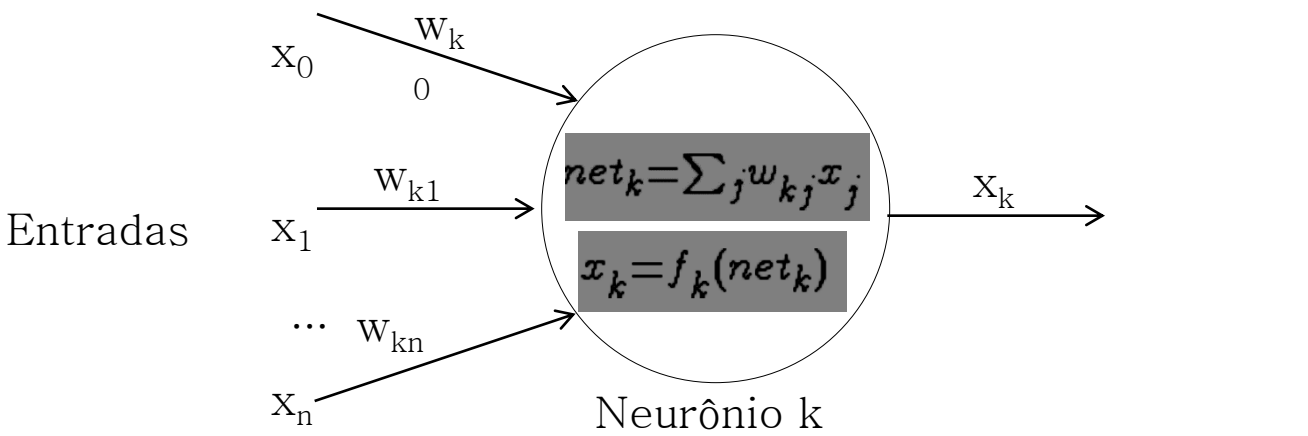


Perceptron



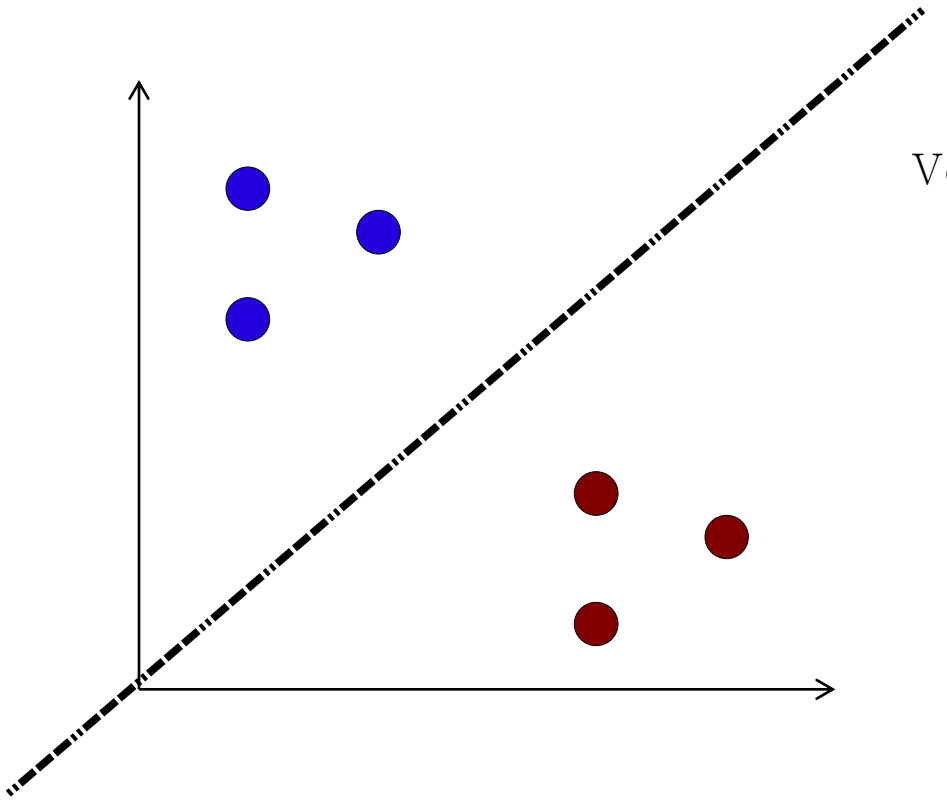
Elemento de Processamento Geral

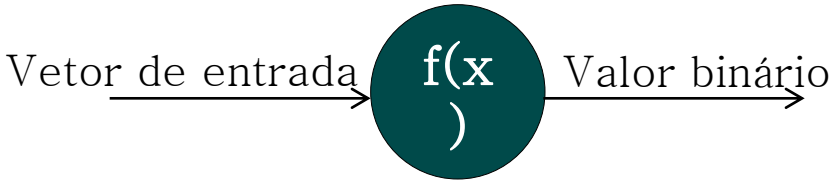
- Neurônios artificiais
  - Nós, unidades ou elementos de processamento
  - Pode receber várias entradas, mas somente produz uma saída
  - Cada conexão para um neurônio está associada a um vetor de pesos **w** (que captura a força da conexão)
  - O aprendizado ocorre com a adaptação de **w**



$f_k(.)$ é chamada função de ativação

Perceptron

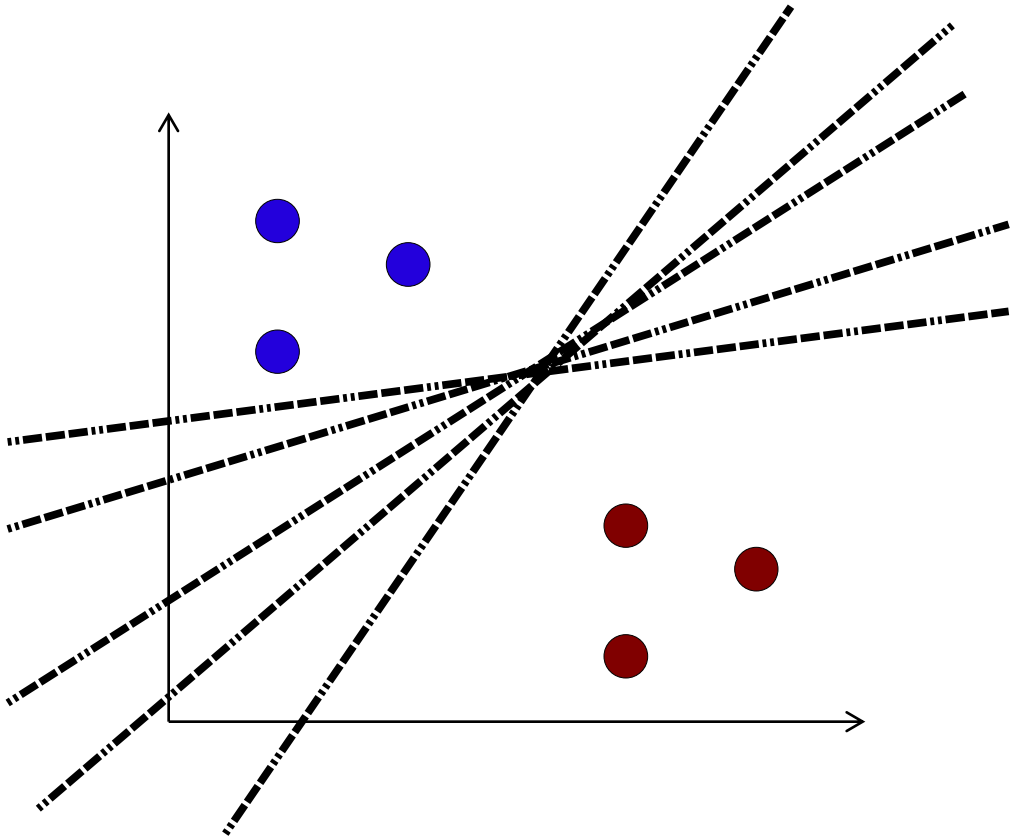




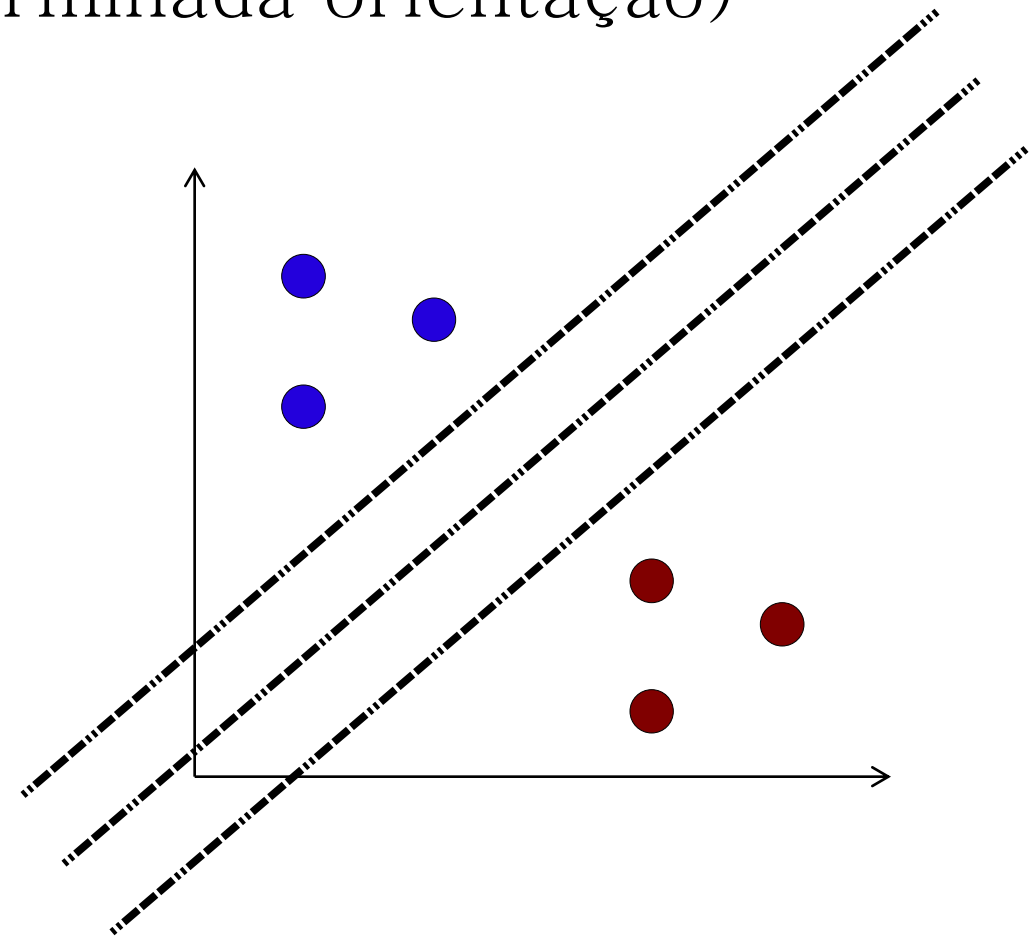
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

w – pesos
x – vetor de entrada
b – bias (termo constante)

$\mathbf{w}$ altera a orientação do hiperplano:



b altera a posição (em relação à origem, para uma determinada orientação)



Perceptron

- Notação usada para descrever o algoritmo:
  - $y_j = f(\mathbf{x}_j)$ é a saída computada pelo perceptron para uma entrada *j*
  - b é o termo de bias
  - $D = \{(\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_s, d_s)\}$ – conjunto de treinamento com *s* instâncias:
    - $\mathbf{x}_j$ é o vetor de entrada com *n* dimensões
    - d_j é a saída desejada (alvo)
  - x_{ji} é o i-ésimo componente do vetor de entrada j
  - w_i é o j-ésimo componente do vetor de pesos
  - α é a taxa de aprendizado (0,1]

Perceptron

- Algoritmo
  - Inicializar o vetor de pesos **w** (*e.g.*, aleatoriamente)
  - Para cada instância j fazer:
    - Computar a saída do perceptron:

$$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$$

    - Adaptar os pesos em função da saída-alvo (d_j):

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}, \text{ for all nodes } 0 \leq i \leq n.$$

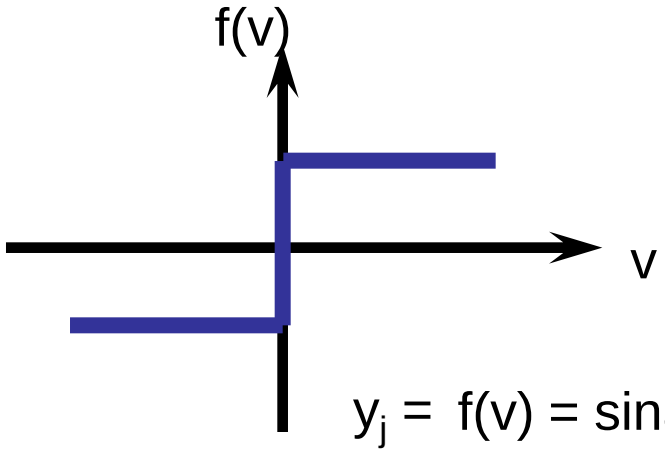
  - Continuar até que um critério de convergência tenha sido atingido (e.g., número de repetições, erro mínimo)

Perceptron

- Com função de limiar

$$v = \sum_{i=0}^{N} x_i w_{ij}$$

$$y_j = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



y_j = f(v) = sinal (v)

Após treinamento:

1 Apresentar padrão **x** *a ser reconhecido*

2 Calcular a saída y

3 Se (y=-1)

Então

x $\in$ *classe 0*

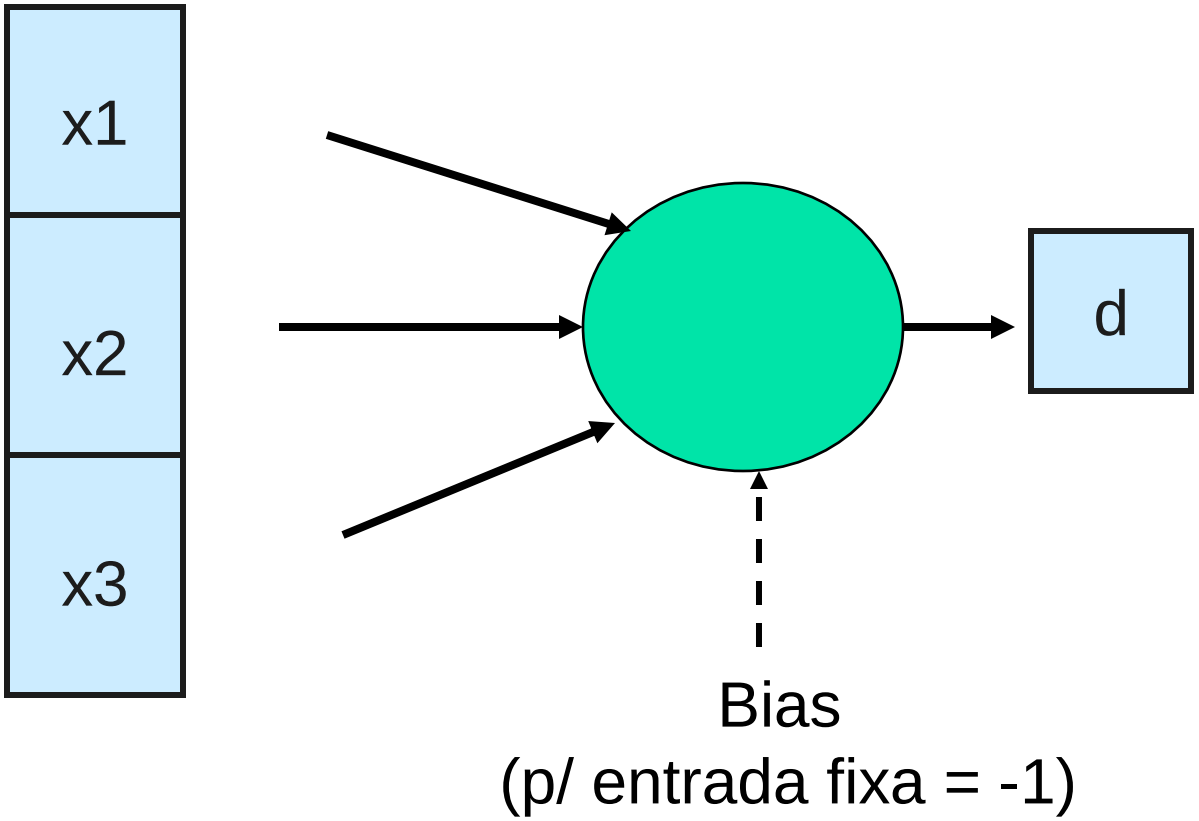
Senão

x $\in$ *classe 1*

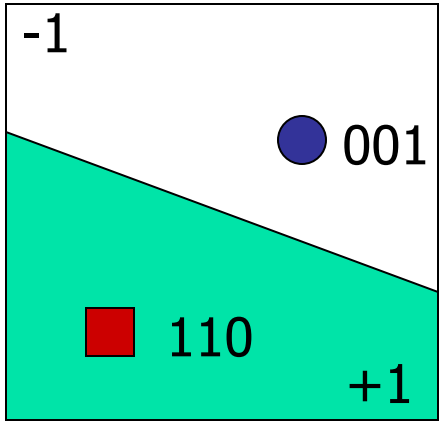
Exemplo

- Perceptron com 3 entradas
- Pesos iniciais $w_0 = 0.4$, $w_1 = -0.6$ e $w_2 = 0.6$
- Bias = 0.5
- Treinar com (001, -1) e (110, +1)
- Utilizar taxa de aprendizado $\eta = 0.4$
- Estimar classes para: [111], [000], [100], [011]

Exemplo



Situação desejada



Exemplo

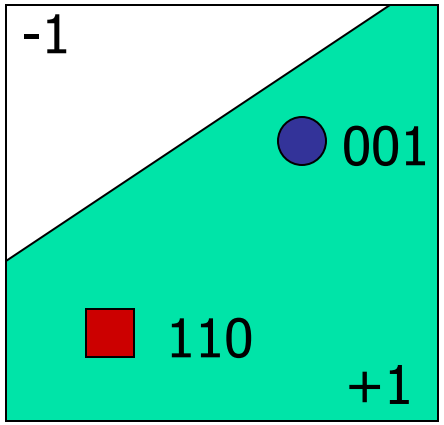
a) Treinando a rede

a.1) Para o padrão 001 (d = -1)

Passo 1: definir a saída da rede

$v = 0(0.4) + 0(-0.6) + 1(0.6) + (-1)(0.5) = 0.1$

$y = +1$ (pois $0.1 \geq 0$)



Passo 2: atualizar pesos

$w_0 = 0.4 + 0.4(0)(-1 - (+1)) = 0.4$

$w_1 = -0.6 + 0.4(0)(-2) = -0.6$

$w_2 = 0.6 + 0.4(1)(-2) = -0.2$

$w_3 = 0.5 + 0.4(-1)(-2) = 1.3$

Exemplo

a.2) Para o padrão 110 (d = +1)

Passo 1: definir a saída da rede

$v = 1(0.4) + 1(-0.6) + 0(-0.2) -1(1.3) = -1.5$

$y = -1$ (pois $-1.5 < 0$)

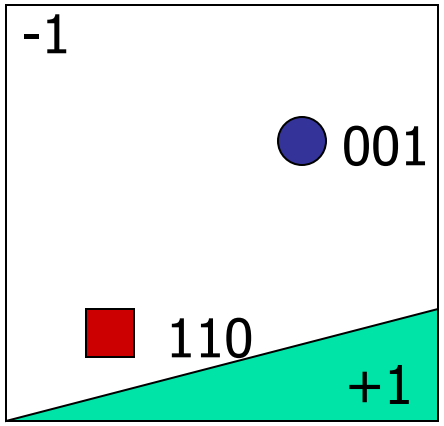
Passo 2: atualizar pesos

$w_0 = 0.4 + 0.4(1)(1 - (-1)) = 1.2$

$w_1 = -0.6 + 0.4(1)(2) = 0.2$

$w_2 = -0.2 + 0.4(0)(2) = -0.2$

$w_3 = 1.3 + 0.4(-1)(2) = 0.5$



Exemplo...

a.3) Para o padrão 001 (d = -1)

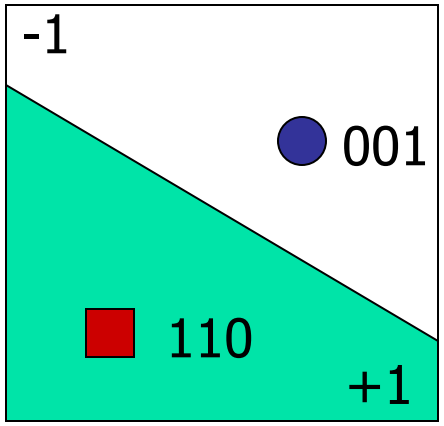
Passo 1: definir a saída da rede

$v = 0(1.2) + 0(0.2) + 1(-0.2) -1(0.5) = -0.7$

$y = -1$ (pois $-0.7 < 0$)

Passo 2: atualizar pesos (d = y)

d = y: pesos não precisam ser modificados



Exemplo...

a.4) Para o padrão 110 (d = +1)

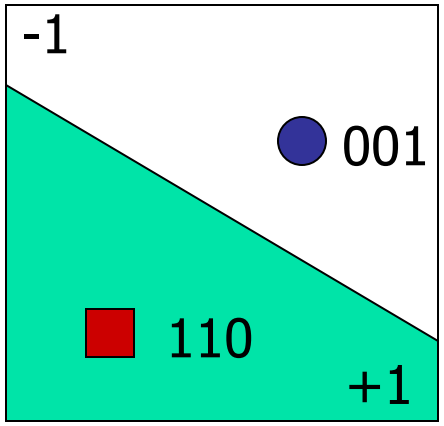
Passo 1: definir a saída da rede

$v = 1(1.2) + 1(0.2) + 0(-0.2) -1(0.5) = +0.7$

$y = +1$ (pois $0.7 > 0$)

Passo 2: atualizar pesos (d = y)

d = y: pesos não precisam ser modificados



Classes para novos vetores:

b.1) Para a instância [111]

$v = 1(1.2) + 1(0.2) + 1(-0.2) -1(0.5) = 0.7$

$y = 1$ (pois $0.7 \geq 0$)) $\Rightarrow$ classe 1

b.2) Para a instância [000]

$v = 0(1.2) + 0(0.2) + 0(-0.2) -1(0.5) = -0.5$

$y = -1$ (pois $-0.5 < 0$) $\Rightarrow$ classe 0

Classes para novos vetores:

b.3) Para a instância [100]

$v = 1(1.2) + 0(0.2) + 0(-0.2) + 1(-0.5) = 0.7$

$y = v = 1$ (pois $0.7 \geq 0$) $\Rightarrow$ classe 1

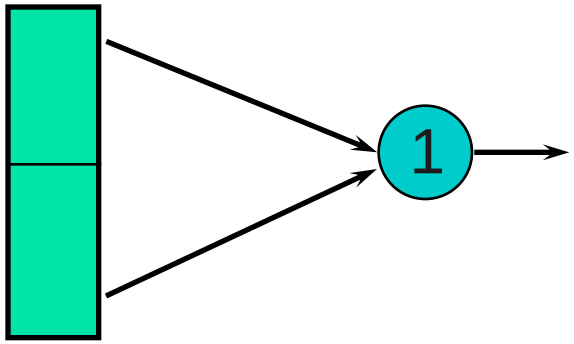
b.4) Para a instância [011]

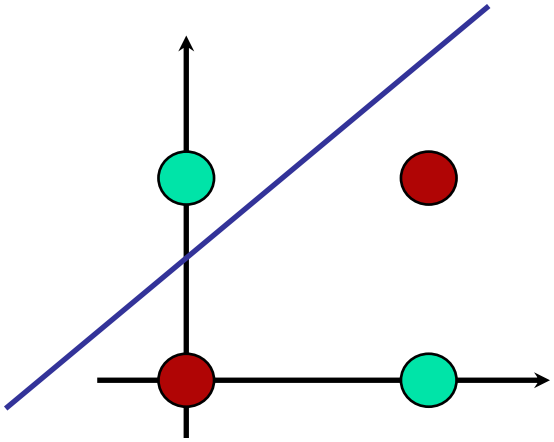
$v = 0(1.2) + 1(0.2) + 1(-0.2) - 1(0.5) = -0.5$

$y = v = -1$ (pois $-0.5 < 0$) $\Rightarrow$ classe 0

Problemas com Perceptron

$$
\begin{aligned}
0, 0 &\rightarrow 0 \\
0, 1 &\rightarrow 1 \\
1, 0 &\rightarrow 1 \\
1, 1 &\rightarrow 0
\end{aligned}
$$

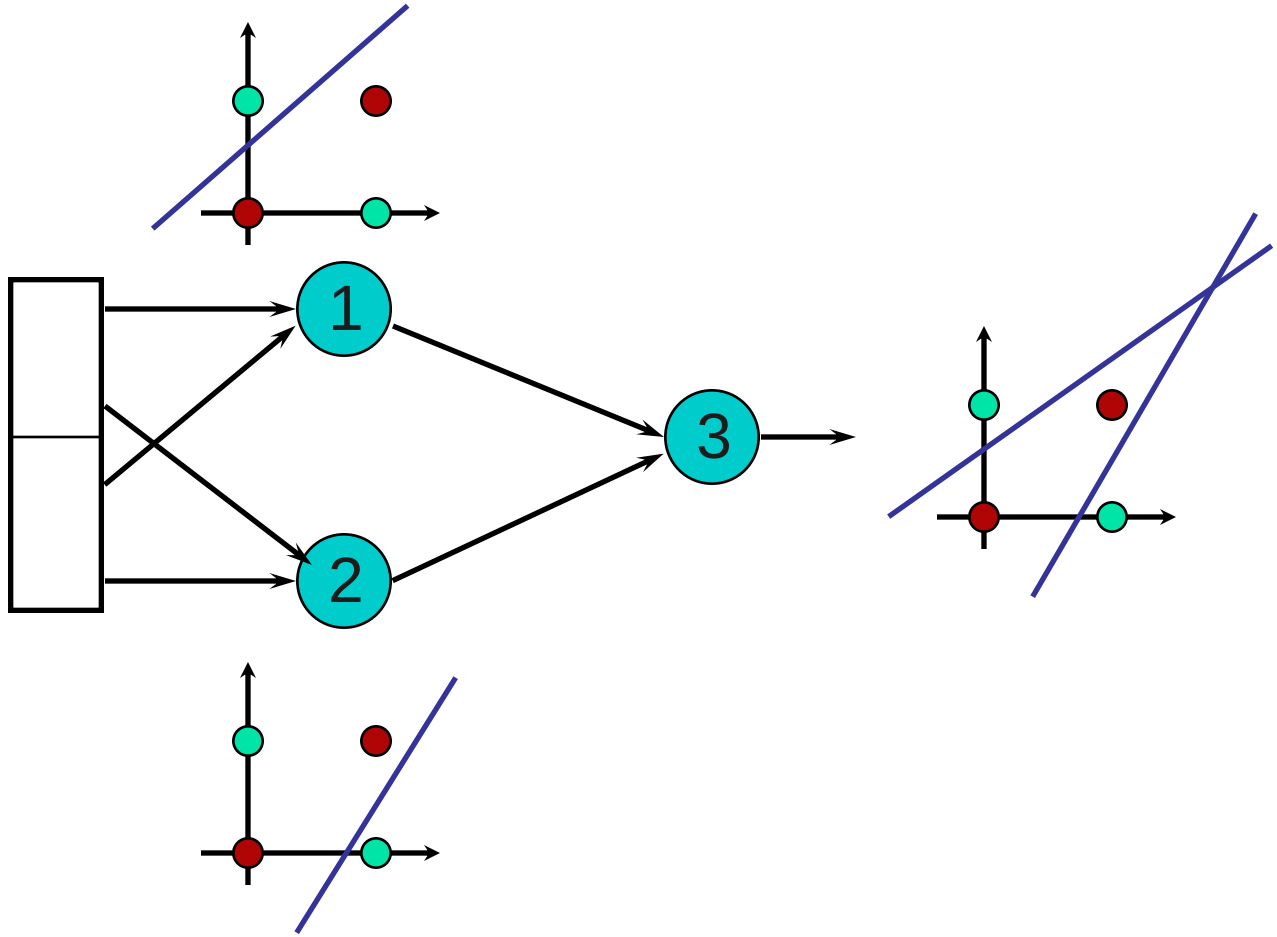




Problemas com Perceptron

- Com uma camada resolve apenas problemas linearmente separáveis
- Grande número de aplicações importantes possuem problemas que não são linearmente separáveis

Solução:
- usar mais de uma camada

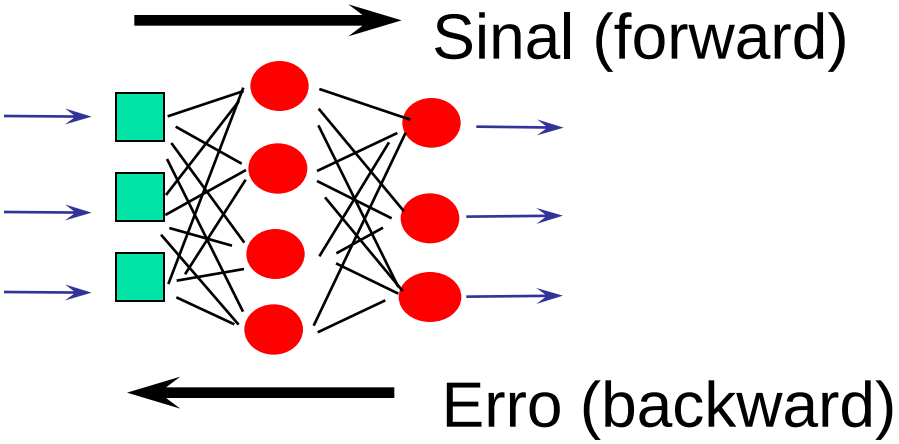


Multi-Layer Perceptron (MLP)

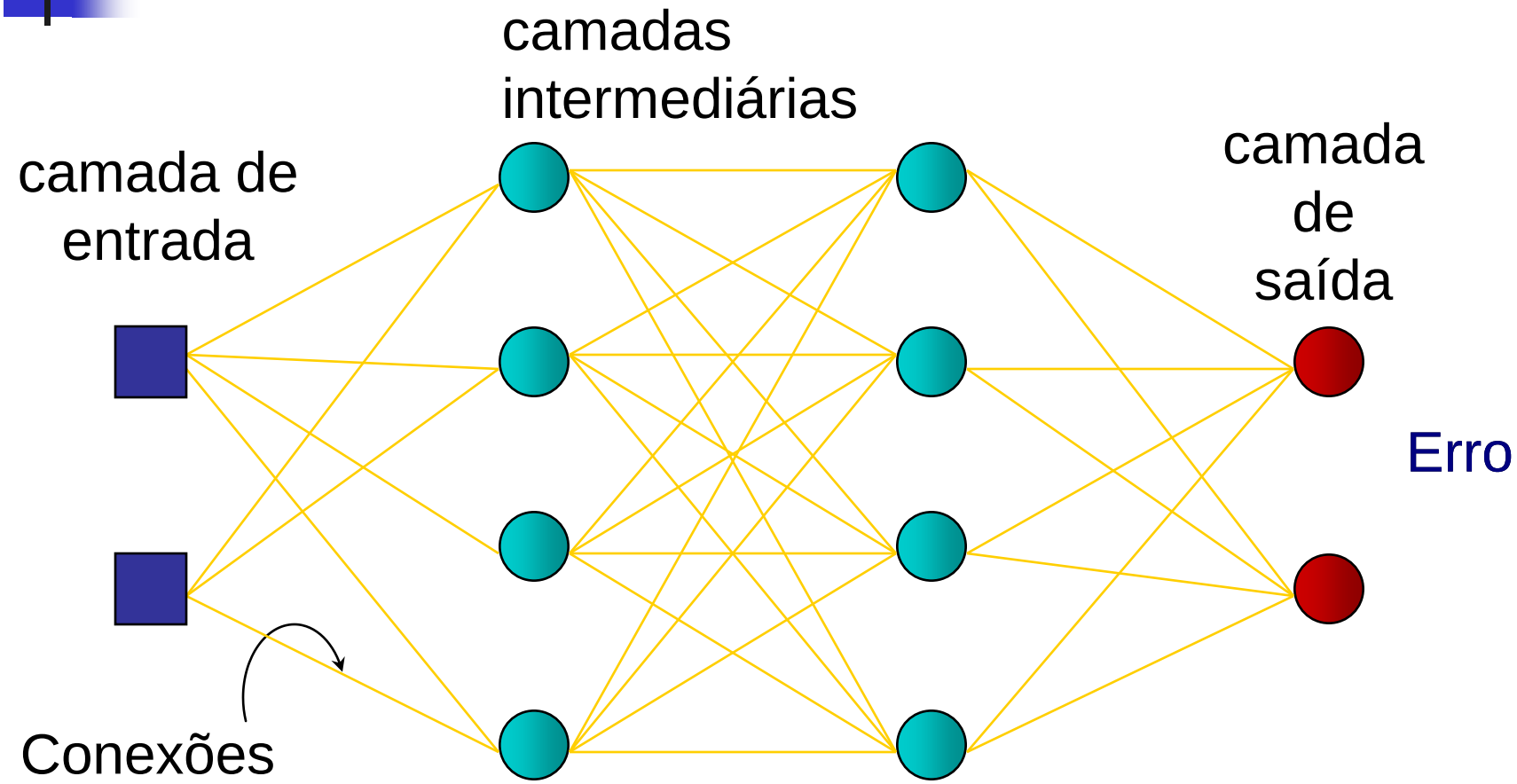
- Arquitetura de RN mais utilizada
  - Possui uma ou mais camadas intermediárias de nós
- Aproximadores universais de funções
  - Uma camada intermediária: qualquer função contínua ou Booleana
  - Duas camadas intermediárias: qualquer função

Backpropagation

- Rede é treinada com pares entrada-saída
- Cada entrada de treinamento é associada a uma saída desejada
- Treinamento em duas fases, cada uma percorrendo a rede em um sentido
  - Fase forward
  - Fase backward



Backpropagation



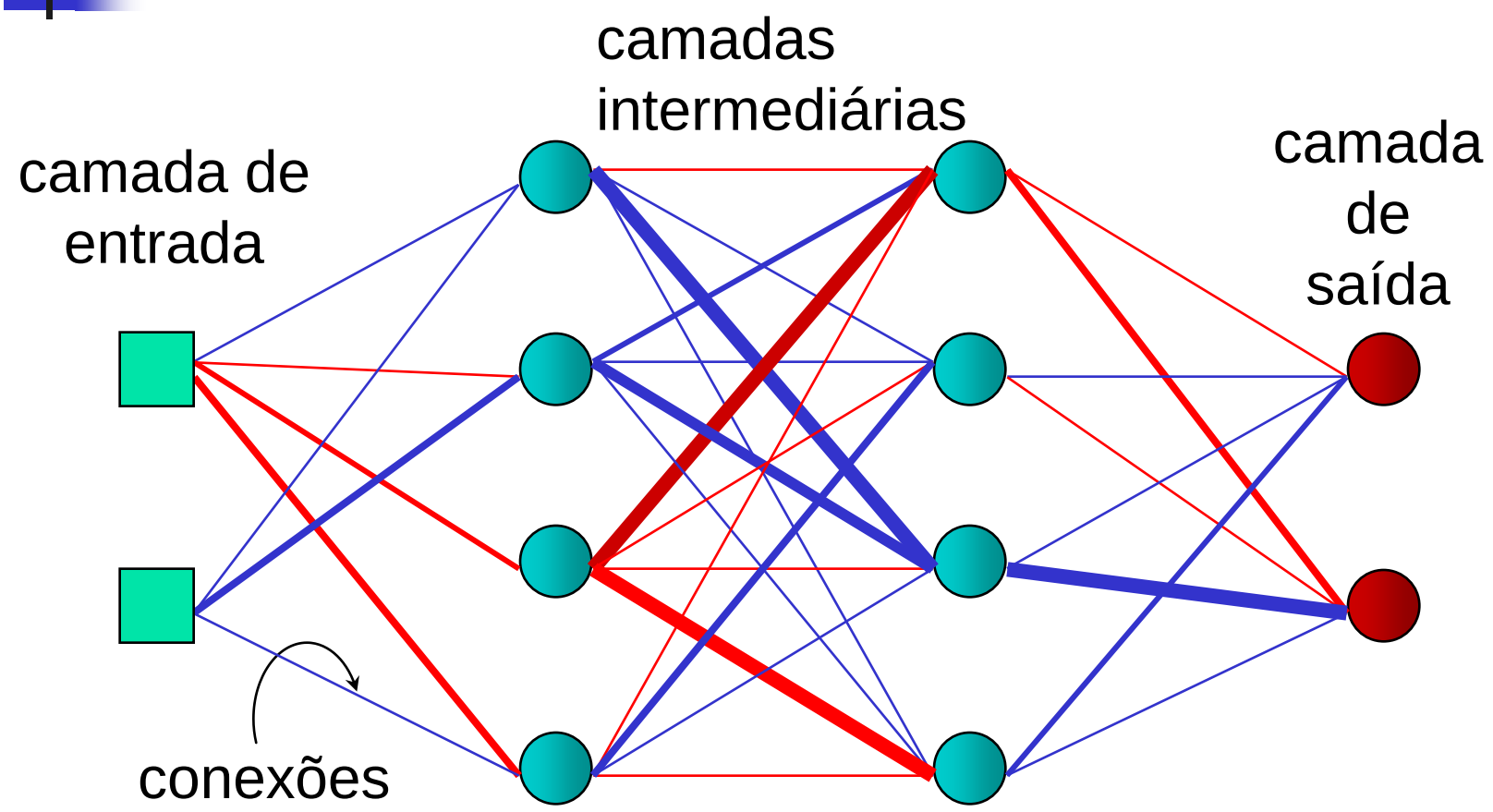
Backpropagation

- Procura reduzir os erros cometidos pela rede, utilizando-os para ajustar os pesos

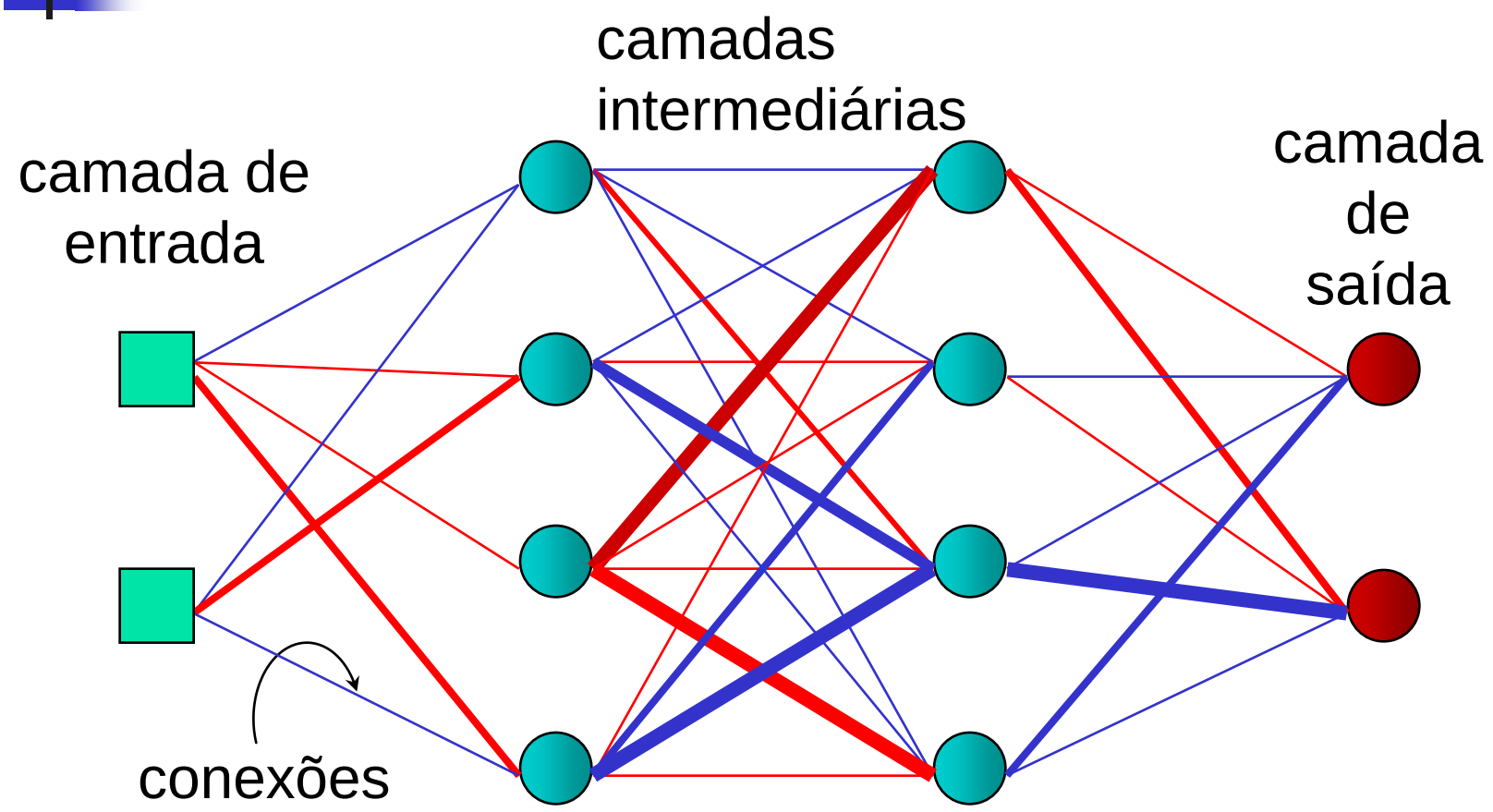
→ Detalhes algorítmicos/matemáticos da técnica de *backpropagation* fogem do escopo deste curso;

→ Haykin, S., Redes Neurais – Princípios e Prática, Bookman.

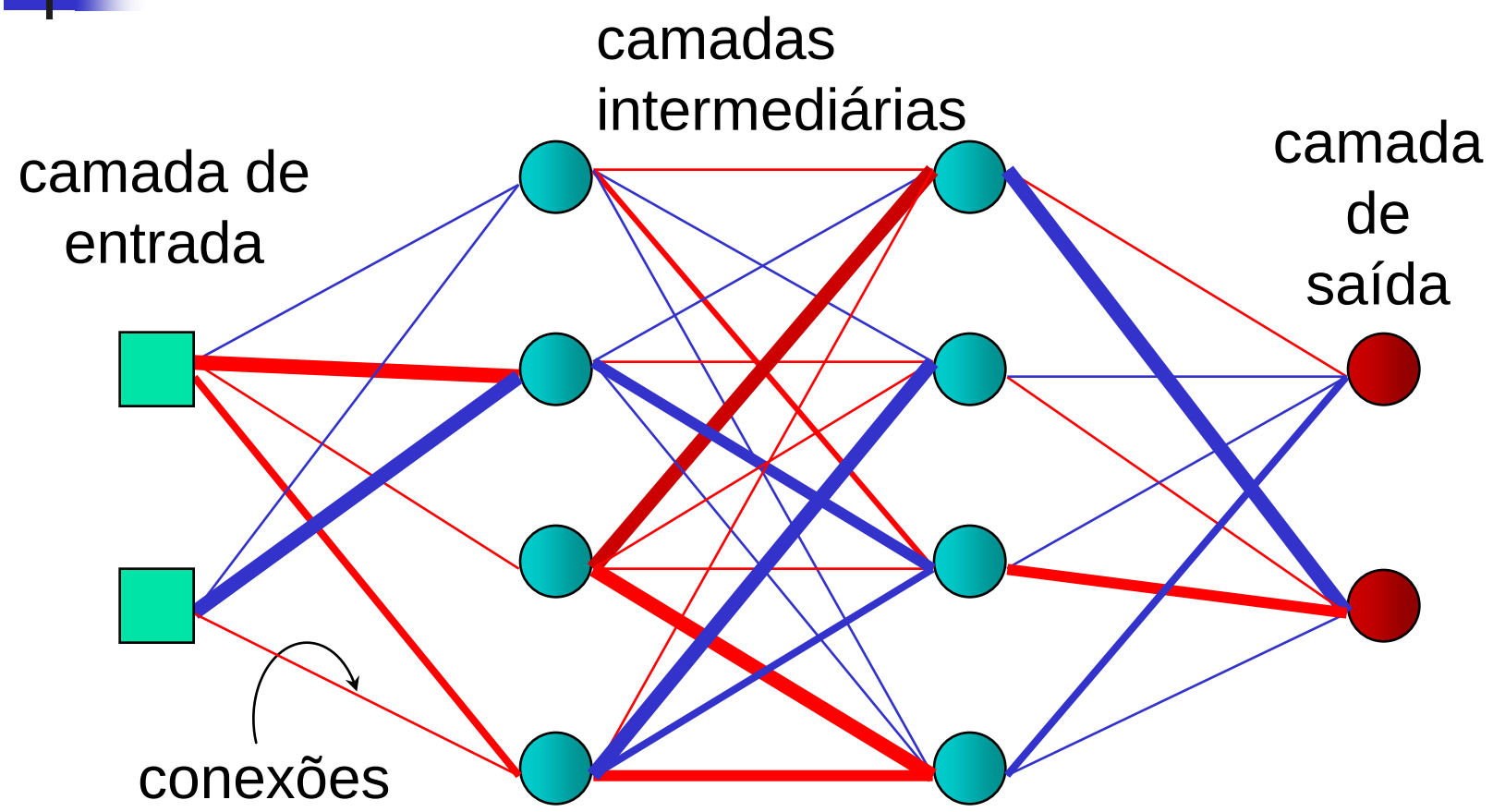
Visualizando o ajuste de pesos



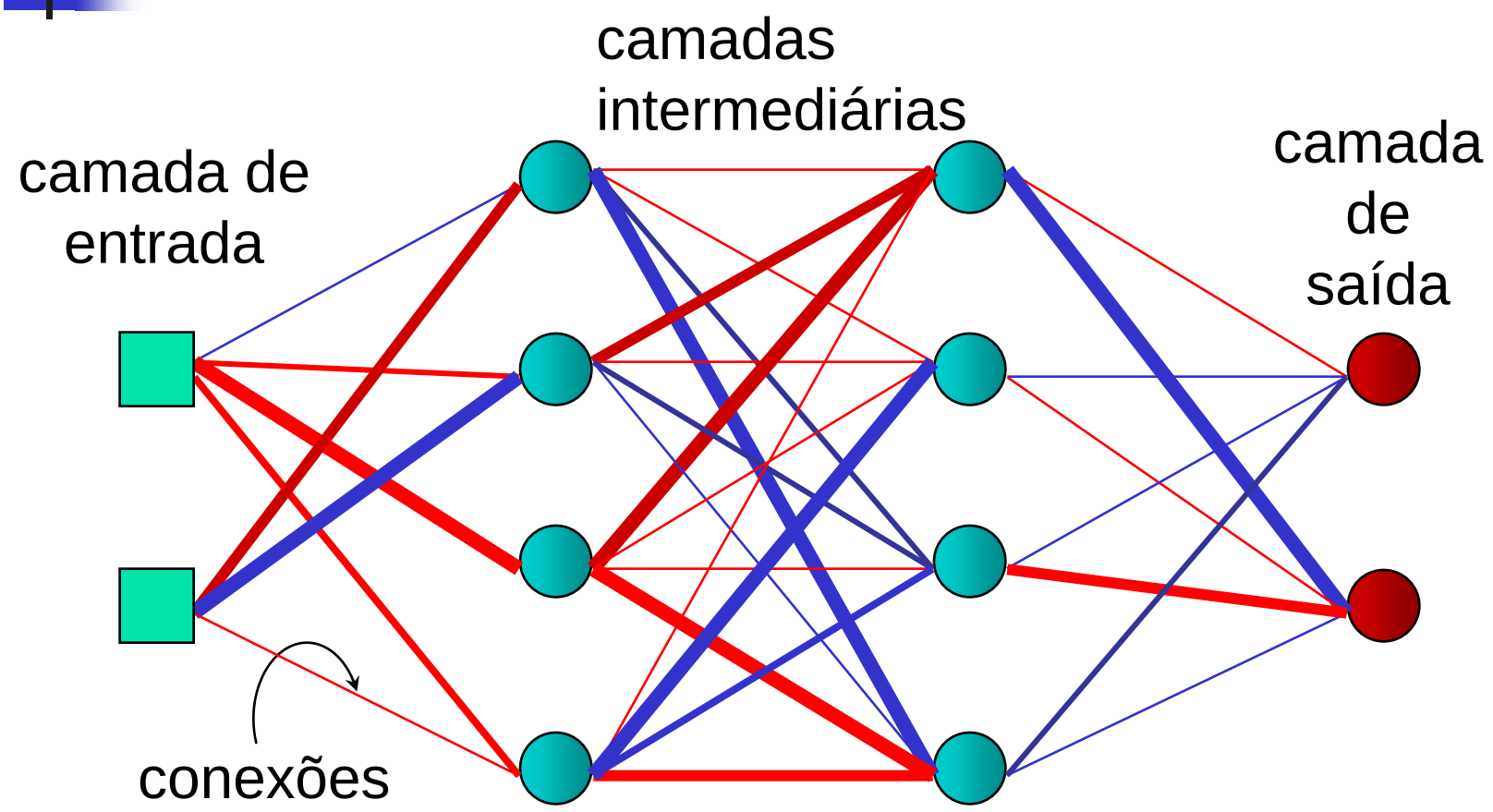
Visualizando o ajuste de pesos



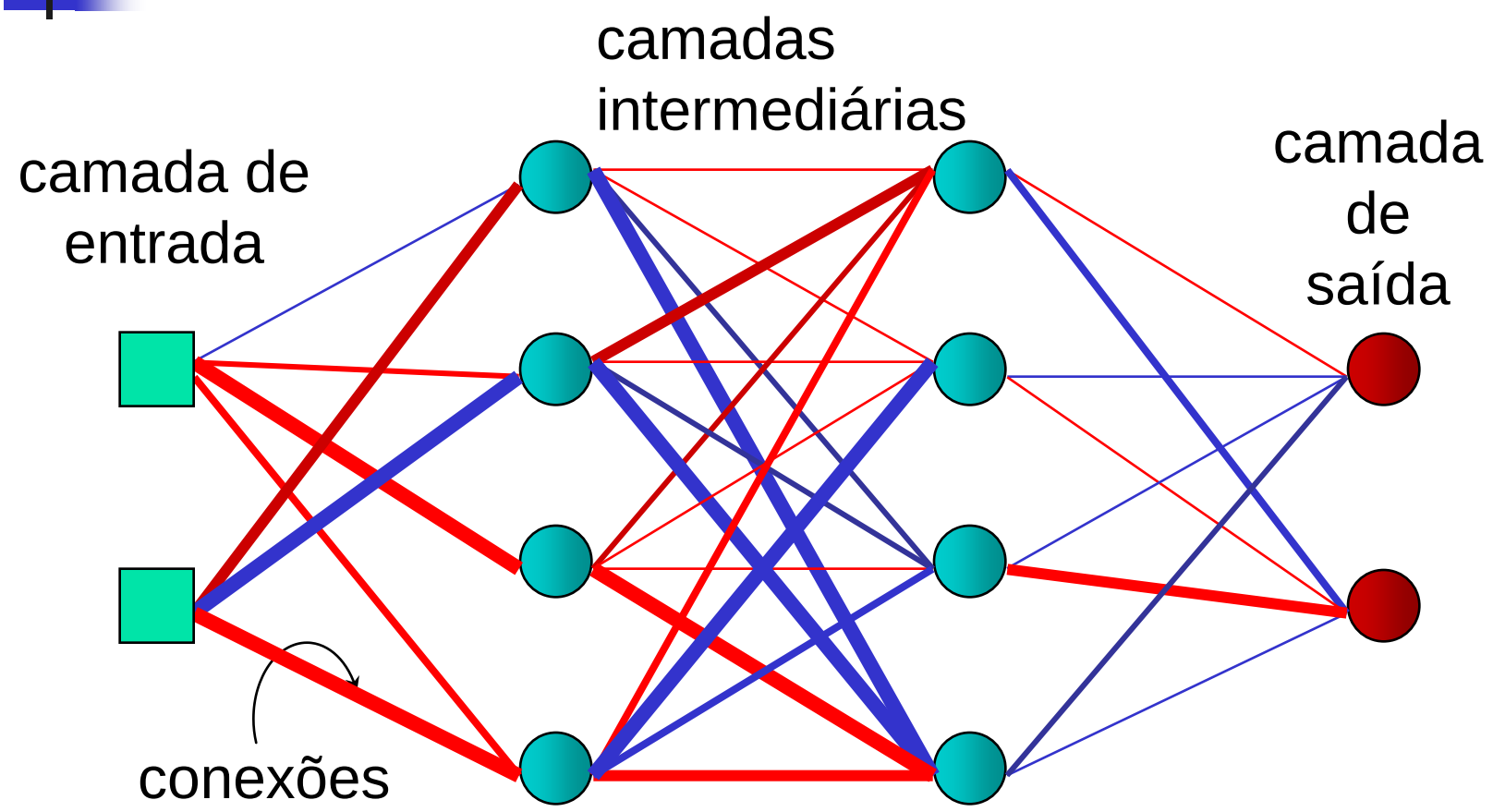
Visualizando o ajuste de pesos



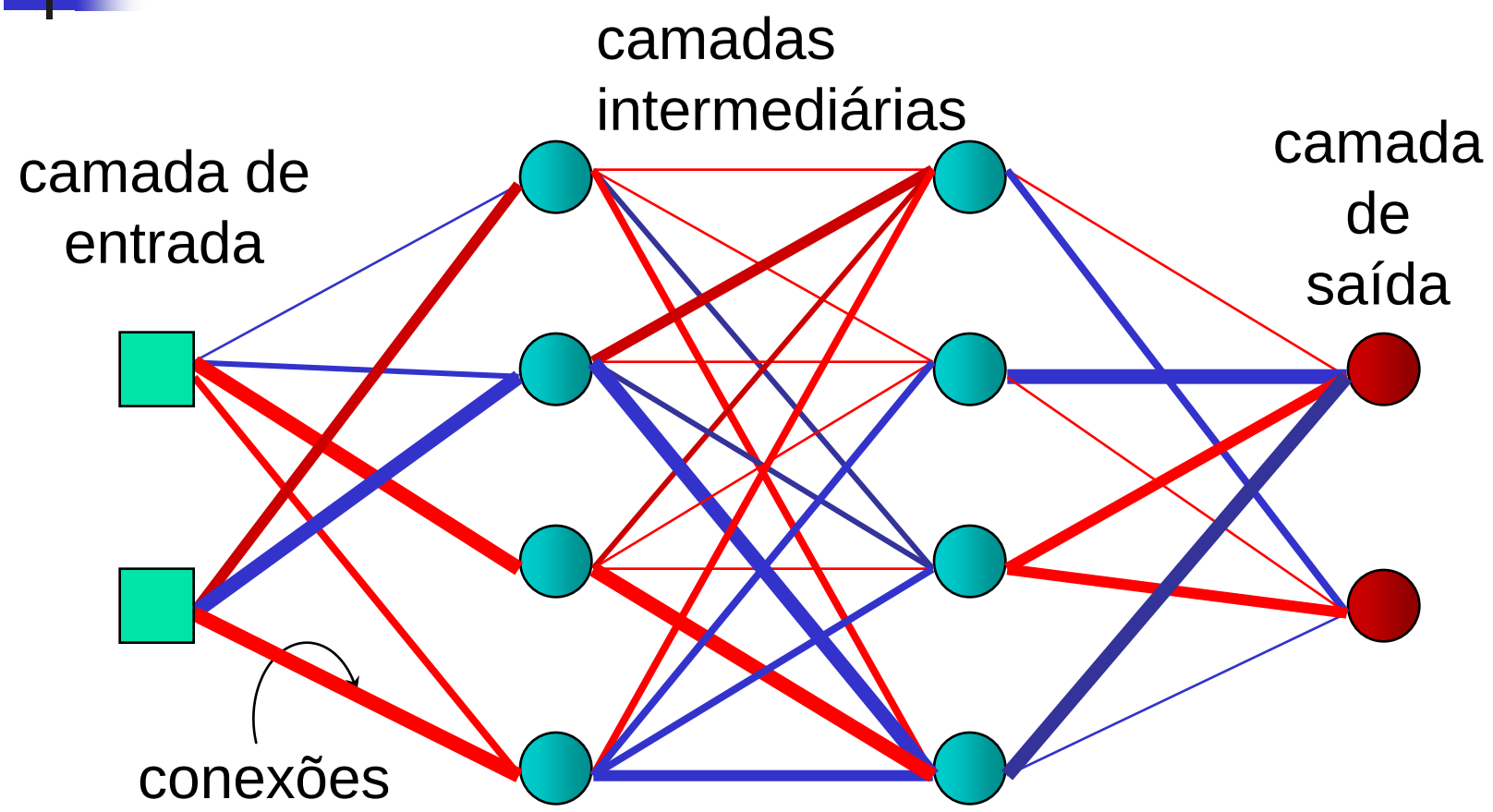
Visualizando o ajuste de pesos



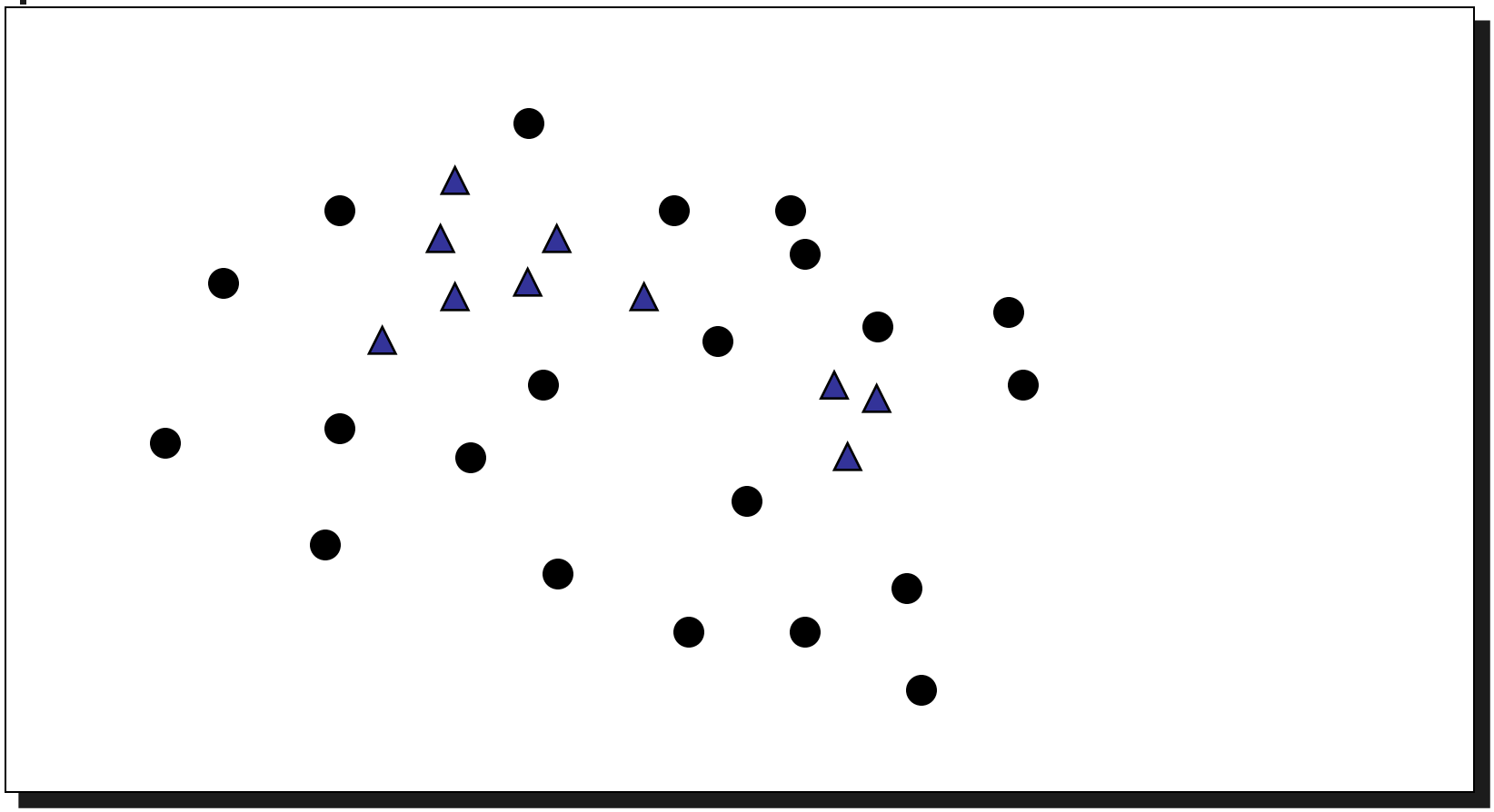
Visualizando o ajuste de pesos



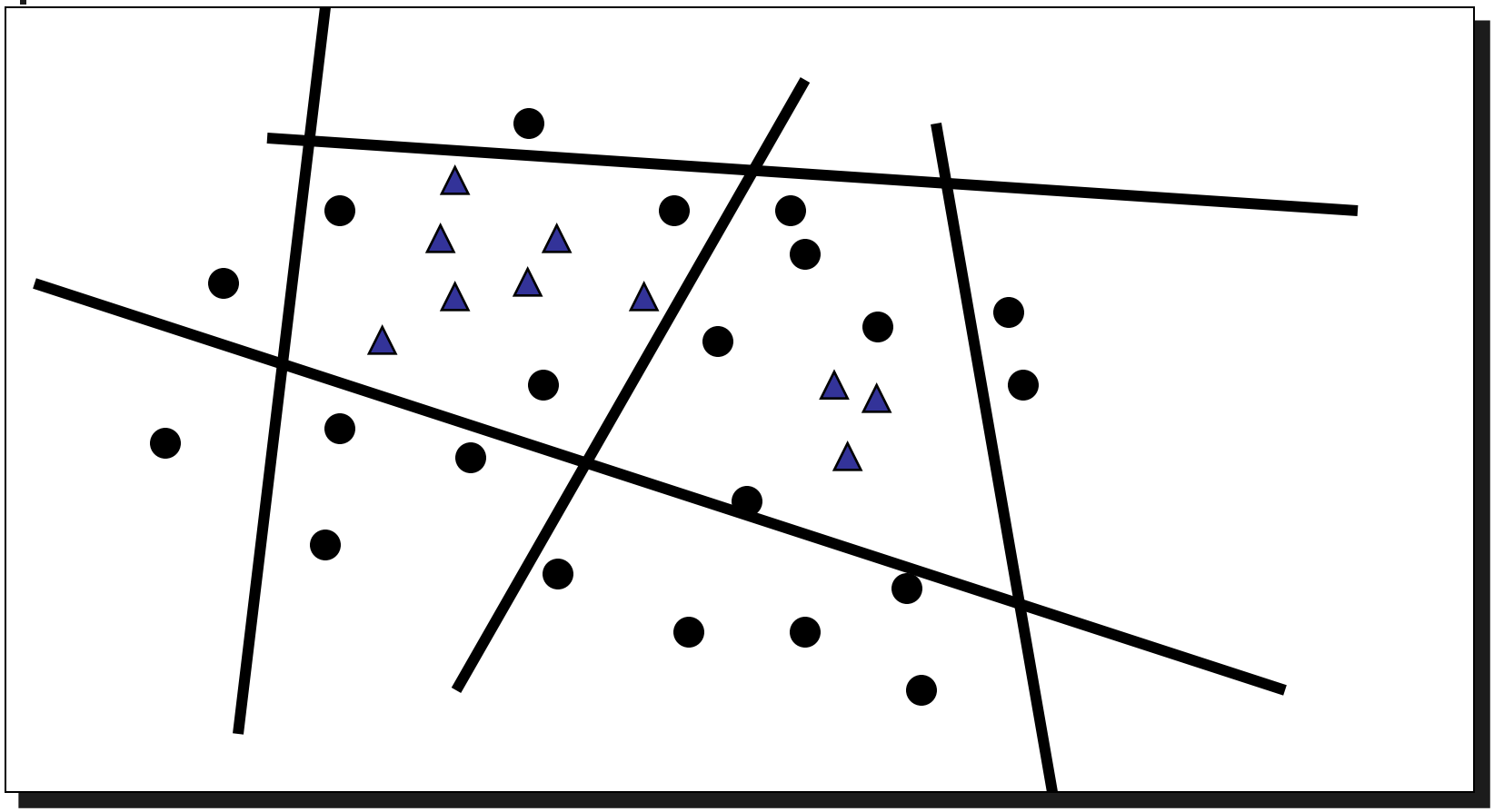
Visualizando o ajuste de pesos



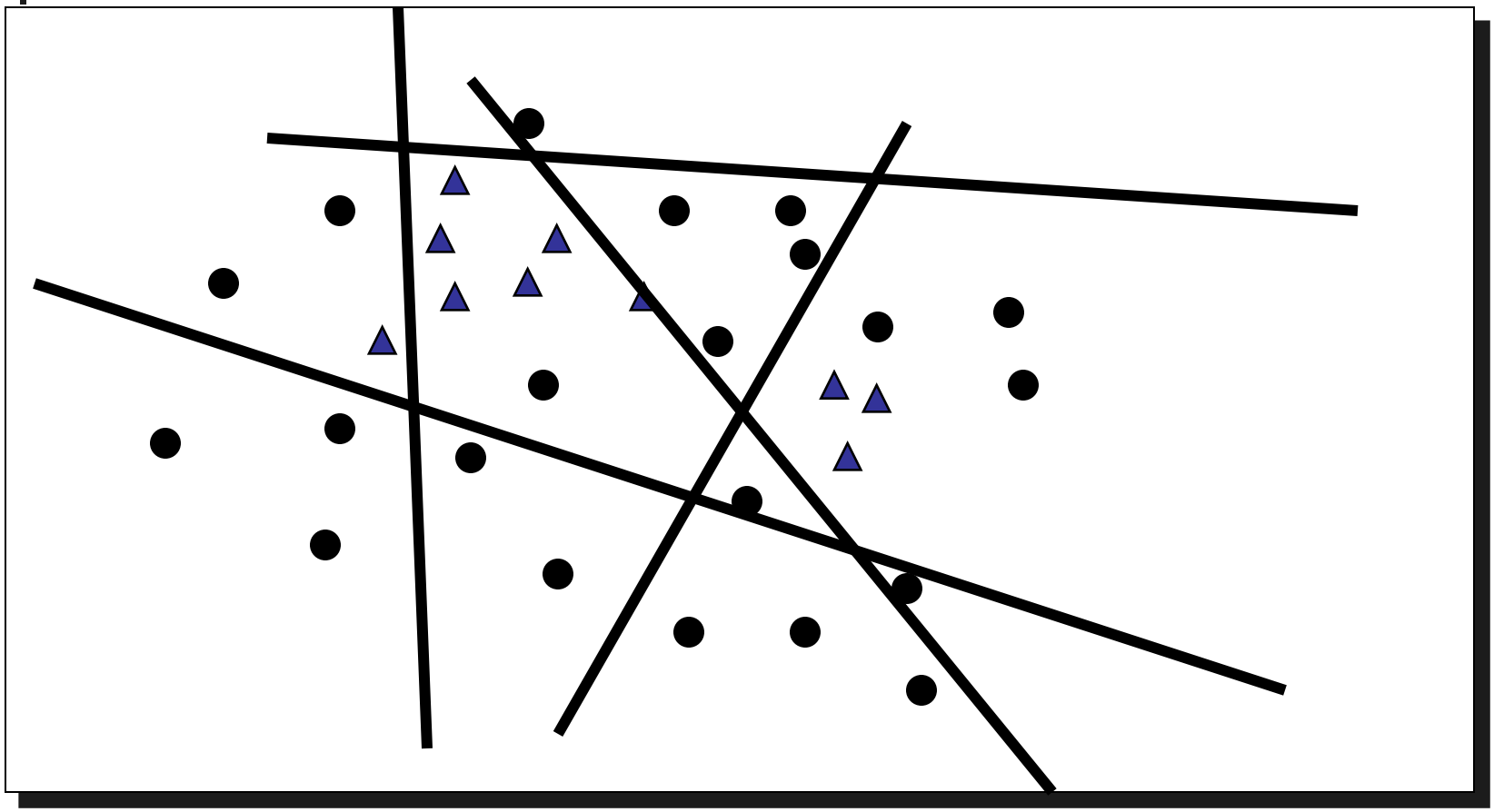
Treinamento



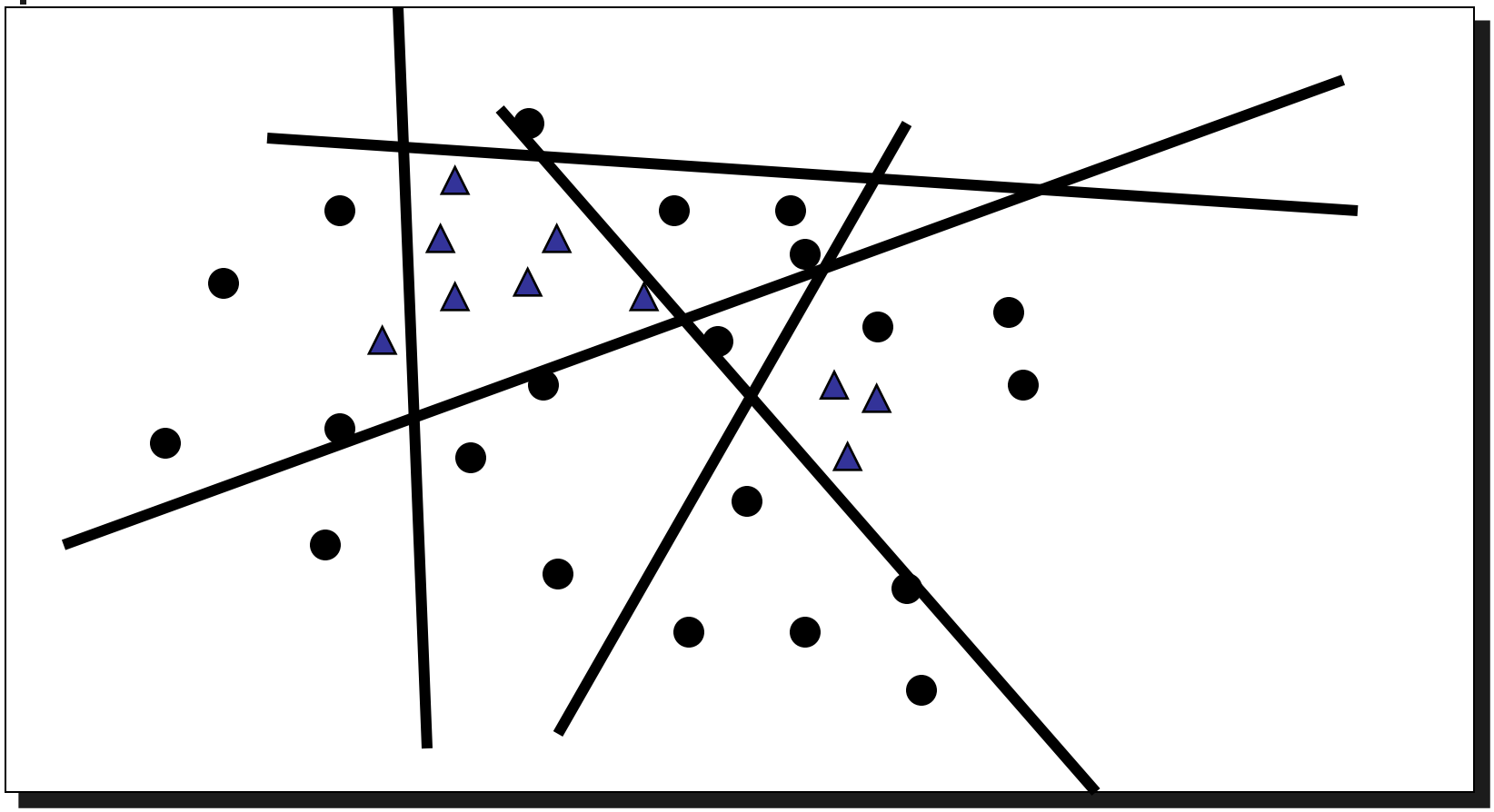
Treinamento



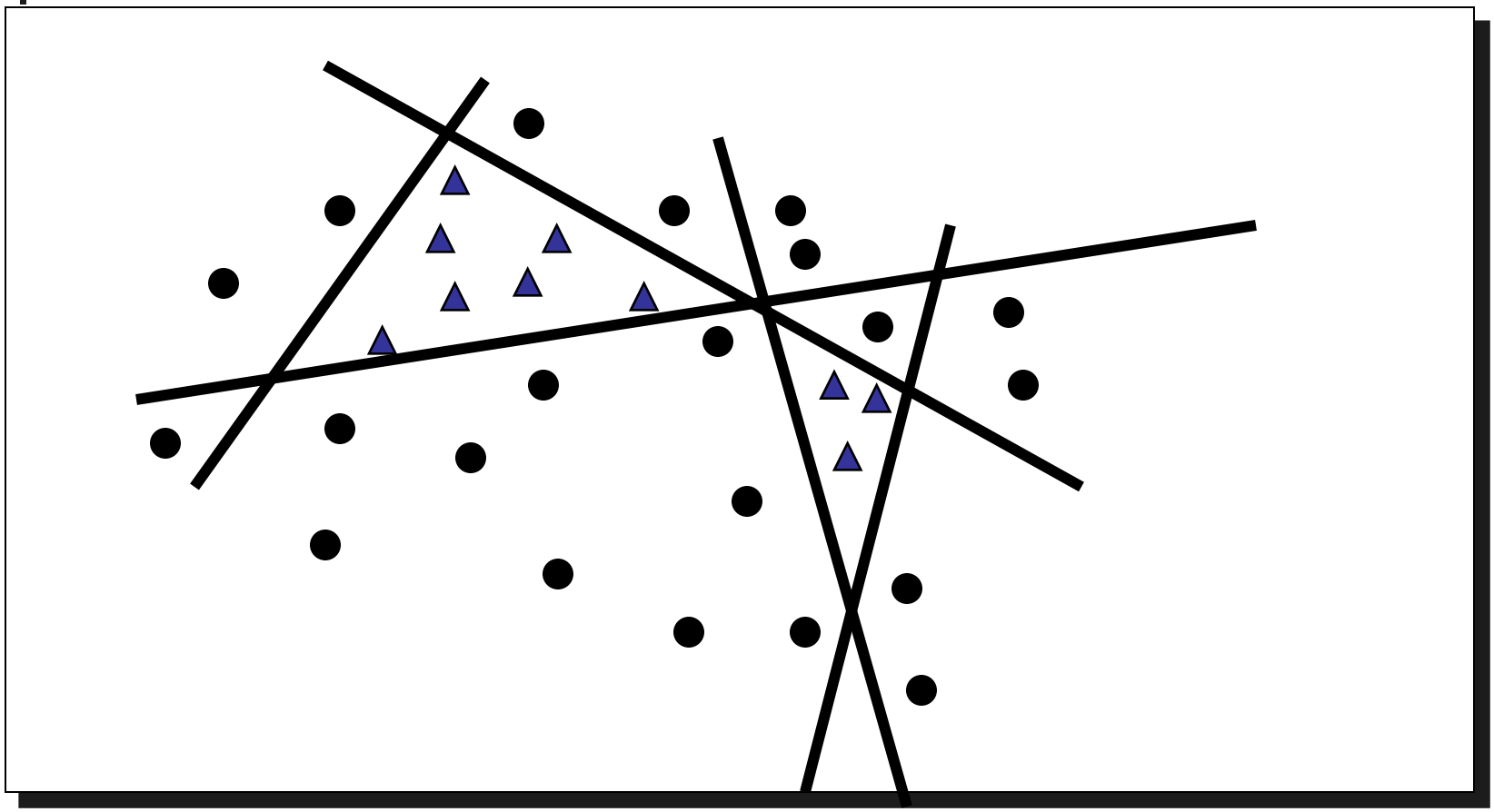
Treinamento



Treinamento



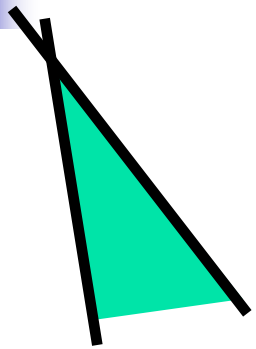
Treinamento



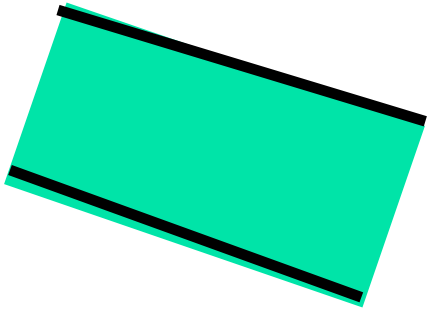
MLPs como classificadores

- Função implementada por cada neurônio
  - Formada pela combinação das funções implementadas por neurônios da camada anterior
    - Camada 1: linhas retas no espaço de decisão
    - Camada 2: regiões convexas
      - Número de lados = número de unidades na camada anterior
    - Camada 3: Combinações de figuras convexas
      - Número de figuras convexas = número de unidades da camada anterior

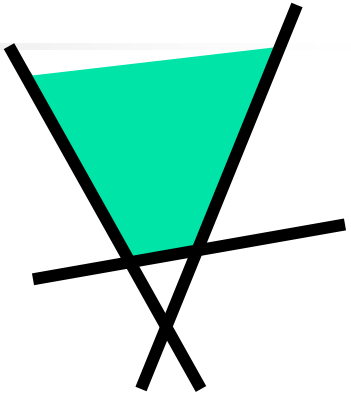
Regiões convexas



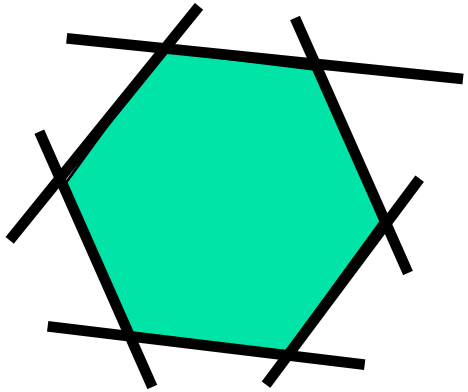
Aberta



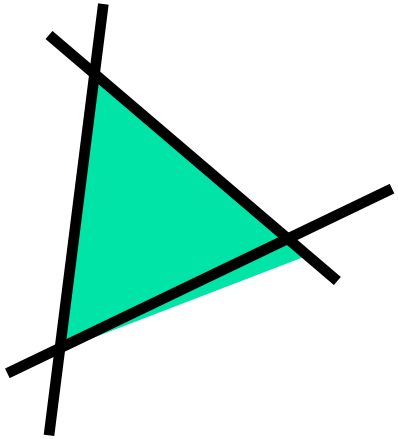
Aberta



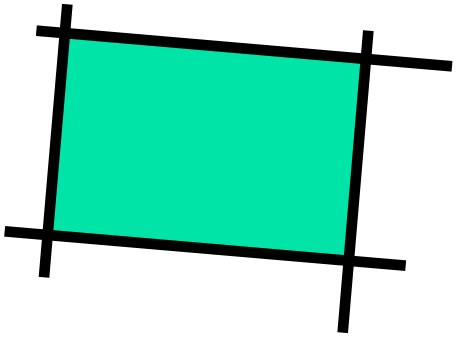
Aberta



Fechada

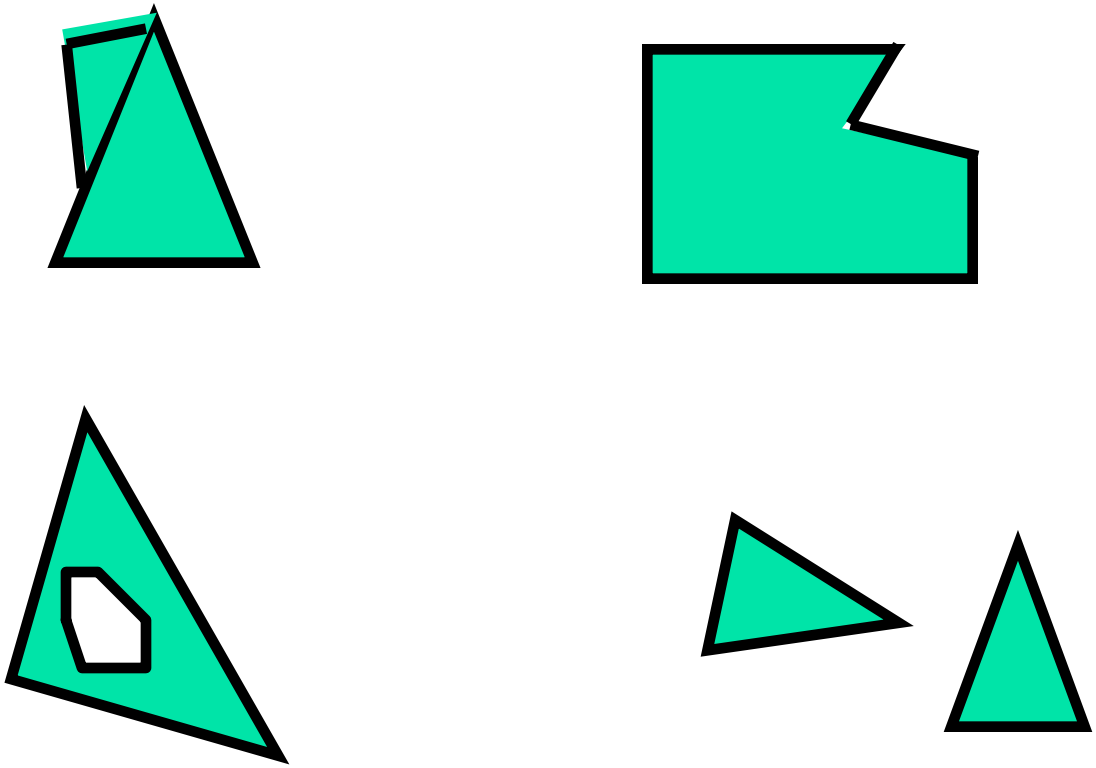


Fechada



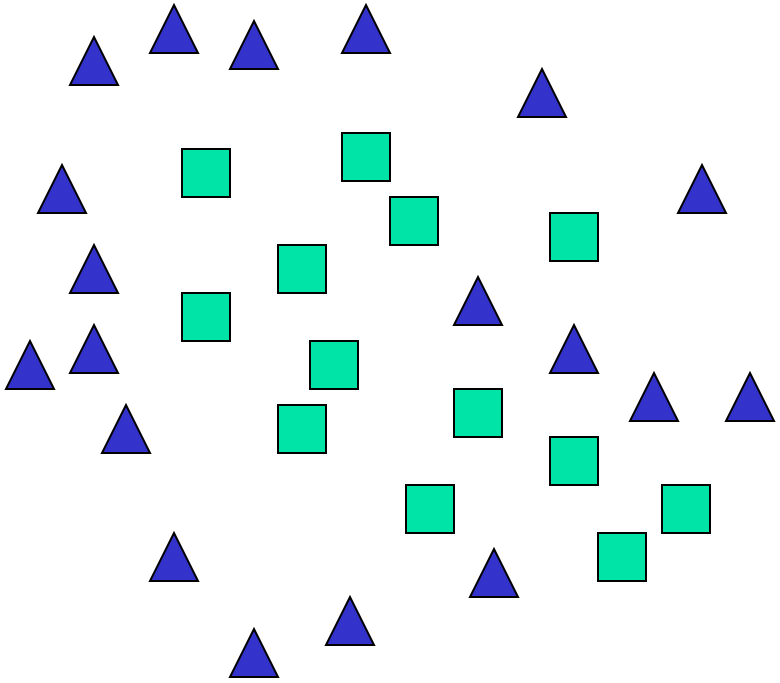
Fechada

Combinando regiões convexas



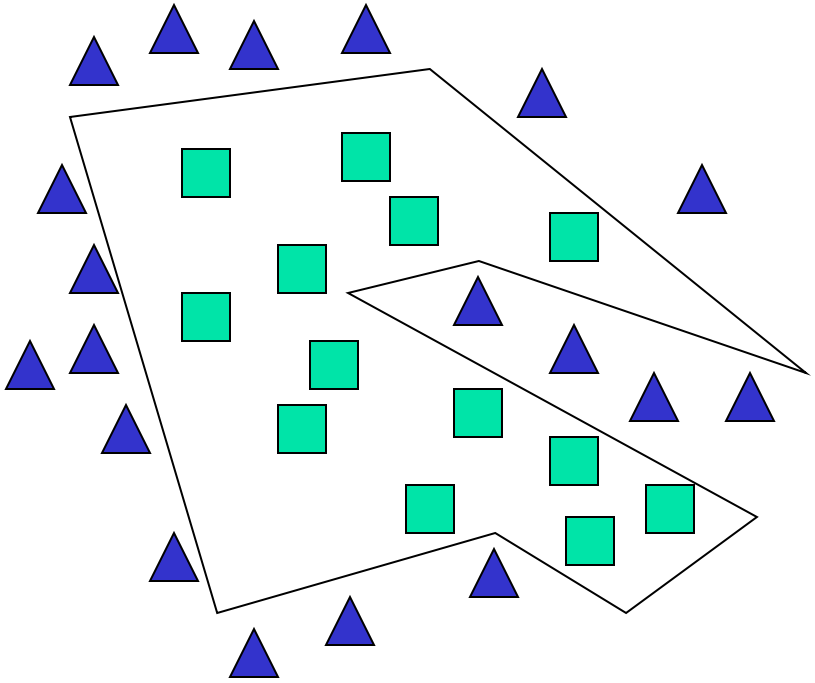
Combinando regiões convexas

- Encontrar fronteiras de decisão que separem os dados abaixo:



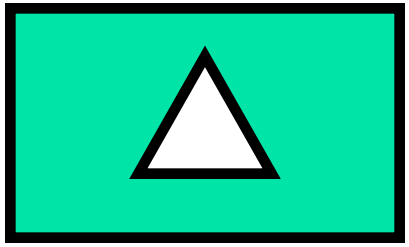
Combinando regiões convexas

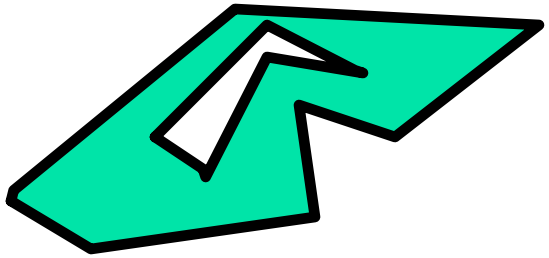
- Encontrar fronteiras de decisão que separem os dados abaixo:



Exercício

- Quantas camadas e pelo menos quantos nodos em cada camada possui a rede que divide o espaço de entradas das formas abaixo:





classe 1
classe 2

classe 1
classe 2

MLPs como classificadores

- Unidades intermediárias
  - Detectores de características
  - Geram uma codificação interna da entrada
  - Dado um número suficientemente grande de unidades intermediárias, é possível formar representações internas para qualquer distribuição dos padrões de entrada

Unidades intermediárias

- Número de camadas intermediárias necessárias
  - 1 camada: pode aproximar qualquer função contínua ou Booleana
  - 2 camadas: pode aproximar qualquer função
  - 3 ou mais camadas: pode facilitar/dificultar o treinamento da rede – questão empírica

Unidades intermediárias

- Número de neurônios por camada
  - Em geral não é conhecido
  - Função do número de entradas e saídas
    - Existem problemas com uma entrada e uma saída que precisam de milhares de unidades
    - Maior parte dos casos: processo de tentativa e erro.

Unidades intermediárias

- Número de neurônios por camada
  - Depende de:
    - Número de exemplos de treinamento
    - Quantidade de ruído
    - Complexidade da função a ser aprendida
      - Distribuição estatística dos dados
  - Pode-se usar rede superdimensionada
    - Interromper o treinamento
    - Controle da norma dos pesos (regularização)

Generalização

- Capacidade de classificar corretamente novos padrões:
  - Não vistos durante o treinamento;
  - Mas da mesma distribuição de probabilidades que gerou a amostra de treinamento (i.i.d)
  - Como estimar?

Dificuldades de aprendizado

- Mínimos locais: solução estável que não é a melhor solução (ótimo global)
  - Incidência pode ser reduzida
    - Empregando taxa de aprendizado decrescente
    - Adicionando nós intermediários
    - Utilizando termo de momentum
    - Adicionando ruído aos dados de treinamento
- Backpropagation é muito lento em superfícies complexas
  - Utilizar métodos de segunda ordem

Dificuldades de aprendizado

- Overfitting
  - Depois de um certo ponto do treinamento, a rede piora ao invés de melhorar
  - Memoriza padrões de treinamento, incluindo suas peculiaridades (piora generalização)
  - Alternativas
    - Encerrar treinamento mais cedo (*early stop*)
    - Reduzir pesos (*weight decay*)

Atualização dos pesos

- Ciclo
  - Apresentação de todos os exemplos de treinamento durante o aprendizado
  - Exemplos devem ser apresentados em ordem aleatória
- Abordagens para atualização dos pesos
  - A cada exemplo (*online*)
  - Por ciclo (*batch*)

Atualização dos pesos

- Por padrão
  - Pesos são atualizados após apresentação de cada exemplo (*gradiente descendente estocástico*)
  - Estável se taxas de aprendizado forem pequenas
    - Taxas elevadas $\Rightarrow$ rede instável
    - Reduzir progressivamente as taxas
  - Mais rápida, principalmente se o conjunto de treinamento for grande e redundante
  - Requer menos memória

Atualização dos pesos

- Por ciclo
  - Pesos atualizados depois que todos os exemplos de treinamento forem apresentados
  - Geralmente mais estável
  - Pode ser lento se o conjunto de treinamento for grande e redundante
  - Estimativa mais precisa do vetor gradiente
- Melhor método depende da aplicação

Variações do backpropagation

- Momentum
- Quickprop
- Newton
- Levenberg Marquardt
- Super Self-Adjusting Backpropagation
- Métodos de gradiente conjugado

Aplicações de RNA

- Reconhecimento de voz e imagem
- Diagnóstico médico
- Controle de robôs
- Controle de processos químicos
- Otimização
- Finanças
- Fraudes
- Fusão de sensores
- Bioinformática

Observações finais

- Redes Neurais têm sido bem sucedidas em problemas de Reconhecimento de Padrões
- Desencorajam o uso de Redes Neurais
  - *Treinamento difícil*
    - Como ajustar seus parâmetros?
  - Caixa preta
    - Como explicar as decisões tomadas pela rede?

Trabalho

- Utilizando bases de dados do repositório de dados da UCI (e.g., IRIS) e o software Weka:
  - Treinar uma rede MLP para classificar dados;
  - Executar validação cruzada;
  - Ajustar arquitetura e parâmetros de aprendizado por tentativa e erro;
  - Reportar os resultados obtidos.

RNs para Aprendizado Não Supervisionado:

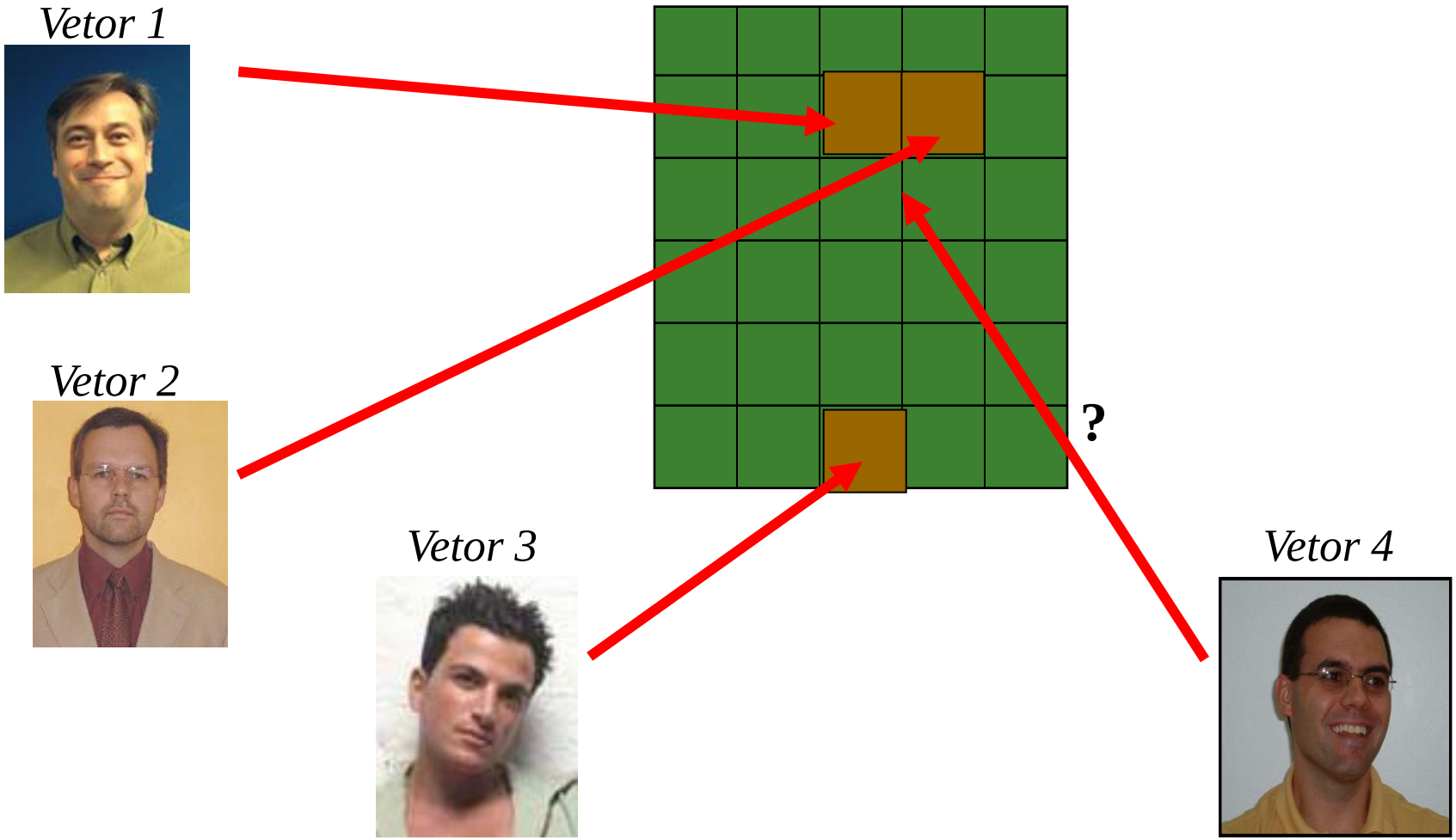
- Aplicações em que não se dispõe de um *supervisor externo:*
  - Reconhecimento de padrões;
  - Extração de características;
  - Agrupamento de Dados (*clustering*);
  - Compressão de dados;
- Modelos *auto-organizáveis* utilizam o paradigma de aprendizado não supervisionado;
- Somente o aprendizado competitivo será abordado;
- Mais especificamente: algoritmo para obter mapas auto organizáveis (SOM) de Kohonen (1990).

Redes SOM:

- Forte inspiração neurofisiológica: baseadas no mapa topológico presente no córtex cerebral;

  $\rightarrow$ Rede biologicamente plausível.
- Cérebro dos animais mais sofisticados possui áreas responsáveis por funções específicas: fala, visão, audição, controle motor, etc;
- Neurônios topologicamente próximos tendem a responder a padrões ou estímulos semelhantes;
- Ordenação topológica é fruto do *feedback* lateral entre células do córtex cerebral;

Noção Intuitiva:



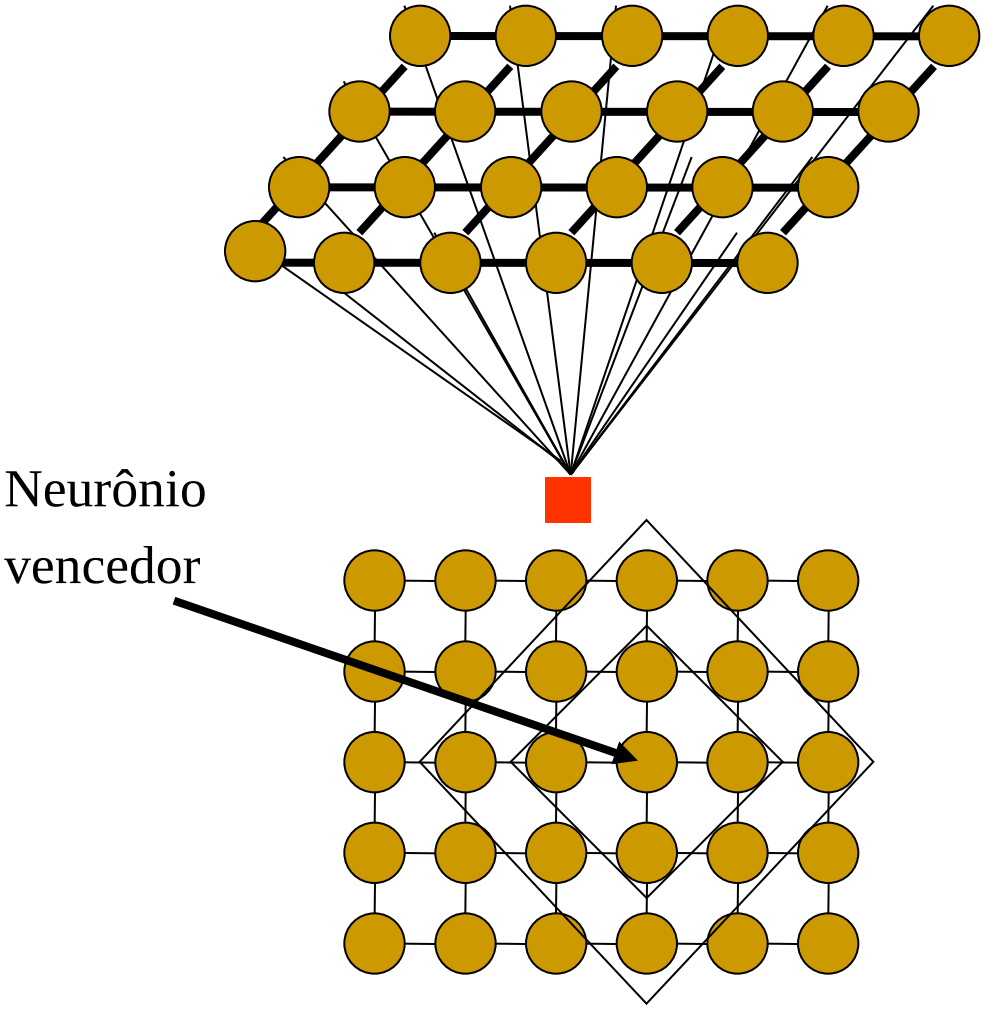
Fonte: Ken McGarry, http://osiris.sunderland.ac.uk/~cs0kmc/

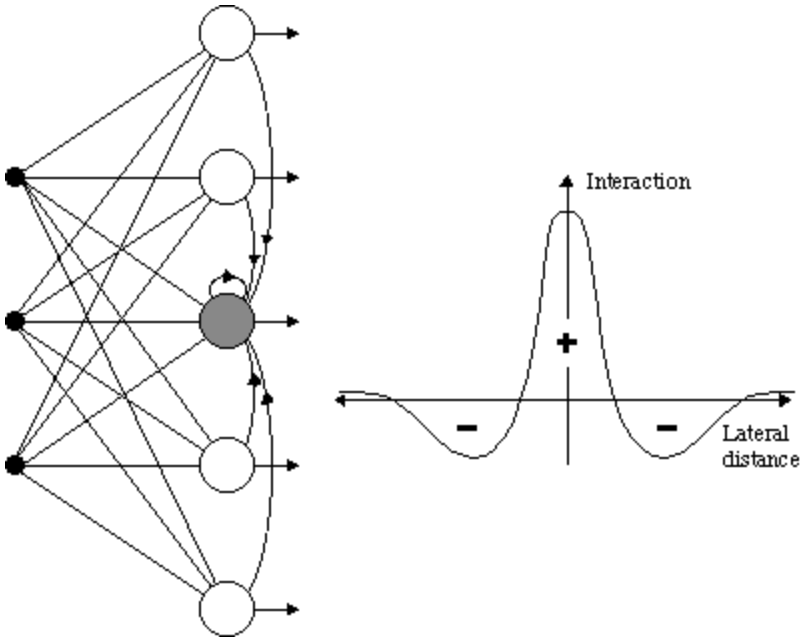
- Funcionamento básico de uma rede SOM:

- Quando um padrão de entrada **p** é apresentado, a rede procura pela unidade mais parecida com **p;**
- Durante o treinamento, a rede aumenta a semelhança da unidade escolhida e de suas vizinhas ao padrão **p**;
- Desta forma, a rede constrói um mapa topológico no qual unidades topologicamente próximas respondem de forma semelhante a padrões de entrada semelhantes;

→ Rede SOM é um modelo do córtex cerebral: seus nodos estão localmente conectados e o processo de adaptação está restrito ao nodo vencedor e seus vizinhos.

Modelos de Kohonen:





Fonte: Ken McGarry, http://osiris.sunderland.ac.uk/~cs0kmc/

Processo competitivo:

Considere um padrão de entrada:

$$\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$$

O vetor sináptico de cada neurônio *j* tem a dimensão do espaço de entrada:

$$\mathbf{w}_j = [w_1, w_2, \ldots, w_m]^T, \quad j=1,\ldots,l, \quad l = \text{número de neurônios.}$$

Para encontrar o *melhor casamento* entre $\mathbf{x}$ e $\mathbf{w}_j$ calcula-se $\mathbf{w}_j^T\mathbf{x}$ para $j=1,\ldots,l$, e escolhe-se *j* correspondente ao maior produto interno, o que é matematicamente equivalente (para vetores normalizados) a determinar o índice $i(\mathbf{x})$ correspondente ao neurônio vencedor *j*:

$$i(\mathbf{x}) = \arg\min_j \left\| \mathbf{x} - \mathbf{w}_j \right\|$$

- Algoritmo básico:

1. Inicializar os pesos (vetores unitários);
2. Retirar um vetor **x** (normalizado) do espaço de entrada;
3. Encontrar o neurônio vencedor;
4. Ajustar os pesos dos neurônios de acordo com:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n).h_{ij(\mathbf{x})}(n).[\mathbf{x}(n) - \mathbf{w}_j(n)]; \quad \text{(normalizar)}$$

5. Se houver modificações significativas no mapa das características, voltar ao passo 2.

Observações:

- Vizinhança em torno do neurônio vencedor *i* decai com a distância lateral: $h_{ij} = f(d_{ij})$;
- A vizinhança $h_{ij(\mathbf{x})}(n)$ e a taxa de aprendizado $\eta(n)$ podem ser ajustados dinamicamente;

Outros tópicos sobre SOM:

- Treinamento em lote: verificar neurônios vencedores para uma apresentação completa dos padrões e depois atualizar os pesos;
- Construção do mapa: muitos neurônios causam super-ajuste e aumento do custo computacional, enquanto que poucos neurônios resultam numa grande variância para os padrões *pertencentes* a cada *cluster*;
  - Iniciar com um número *sub-estimado* de neurônios;
  - Treinar a rede para uma amostra dos padrões entrada;
  - Encontrar o neurônio *N* que apresenta o maior erro de quantização (*Learning Vector Quantizer – LVQ)*, inserindo uma linha e uma coluna na sua vizinhança;
  - Interromper o crescimento do mapa quando o erro de quantização for menor do que o tolerado ou quando se tiver atingido um tamanho máximo especificado para o mapa;

Exercício: treinar uma rede SOM (grade com 4 neurônios) para a base de dados da tabela abaixo (m = 2), considerando que a vizinhança é representada pelos dois neurônios mais próximos e que a taxa de aprendizado é fixa e igual a 0.5. Assuma que os vetores de pesos iniciais sejam todos iguais a [5,5] e que uma época é suficiente para o treinamento. Apresente os vetores à rede de acordo com a ordem em que os mesmos aparecem na tabela:

x_1	x_2	x'_1	x'_2
2	3	*0.55*	*0.83*
1	2	*0.45*	*0.89*
3	2	*0.83*	*0.55*
7	8	*0.66*	*0.75*
6	9	*0.55*	*0.83*
10	9	*0.74*	*0.67*

Solução:

Atualizar pesos de acordo com: $\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + 0.5[\mathbf{x}(n) - \mathbf{w}_j(n)]$.