Redes Neurais

Prof. Lucas Cambuim

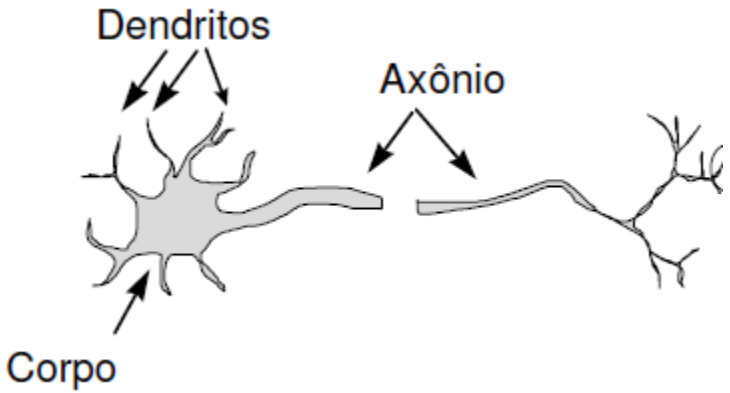
Redes Neurais Artificiais (RNA)

Redes Neurais Artificiais (RNA)

São sistemas inspirados nos neurônios biológicos e na **estrutura maciçamente paralela** do cérebro, com capacidade de **adquirir, armazenar** e **utilizar conhecimento experimental**.

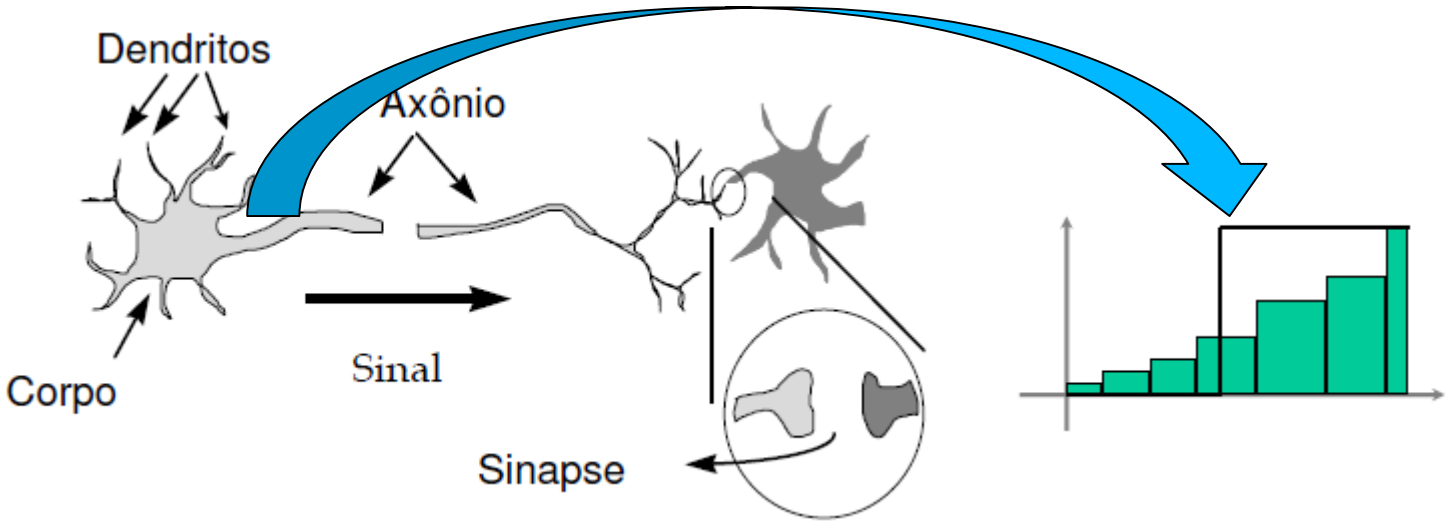
Inspiração biológica

- O sistema nervoso é formado por um conjunto extremamente complexo de células, **os neurônios**. Eles têm um papel essencial na determinação do funcionamento e comportamento do corpo humano e do raciocínio. Os neurônios são formados pelos dendritos, que são um conjunto de terminais de entrada, pelo corpo central, e pelos axônios que são longos terminais de saída.



Inspiração biológica

- Os neurônios se comunicam através de sinapses. Sinapse é a região onde dois neurônios entram em contato e através da qual os impulsos nervosos são transmitidos entre eles. Os impulsos recebidos por um neurônio A, em um determinado momento, são processados, e **atingindo um dado limiar de ação**, o neurônio A dispara, produzindo uma substância neurotransmissora que flui do corpo celular para o axônio, que pode estar conectado a um dendrito de um outro neurônio B. O neurotransmissor pode diminuir ou aumentar a polaridade da membrana pós-sináptica, inibindo ou excitando a geração dos pulsos no neurônio B. Este processo depende de vários fatores, como a geometria da sinapse e o tipo de neurotransmissor.



Fatos Curiosos

Há cerca de 100 bilhões deles no cérebro e na coluna vertebral.

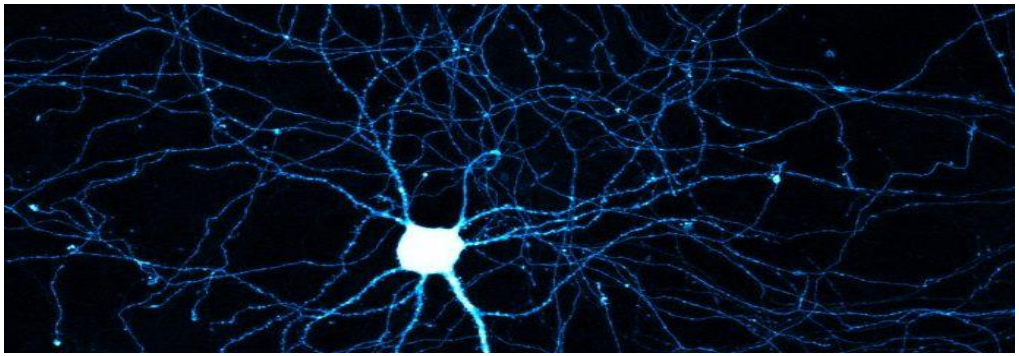
Cada neurônio tem cerca de 10.000 sinapses com outros neurônios.

A maioria deles está localizado no córtex cerebral.

O córtex existe apenas nos cérebros de mamíferos.

O córtex é identificado popularmente como massa cinzenta.

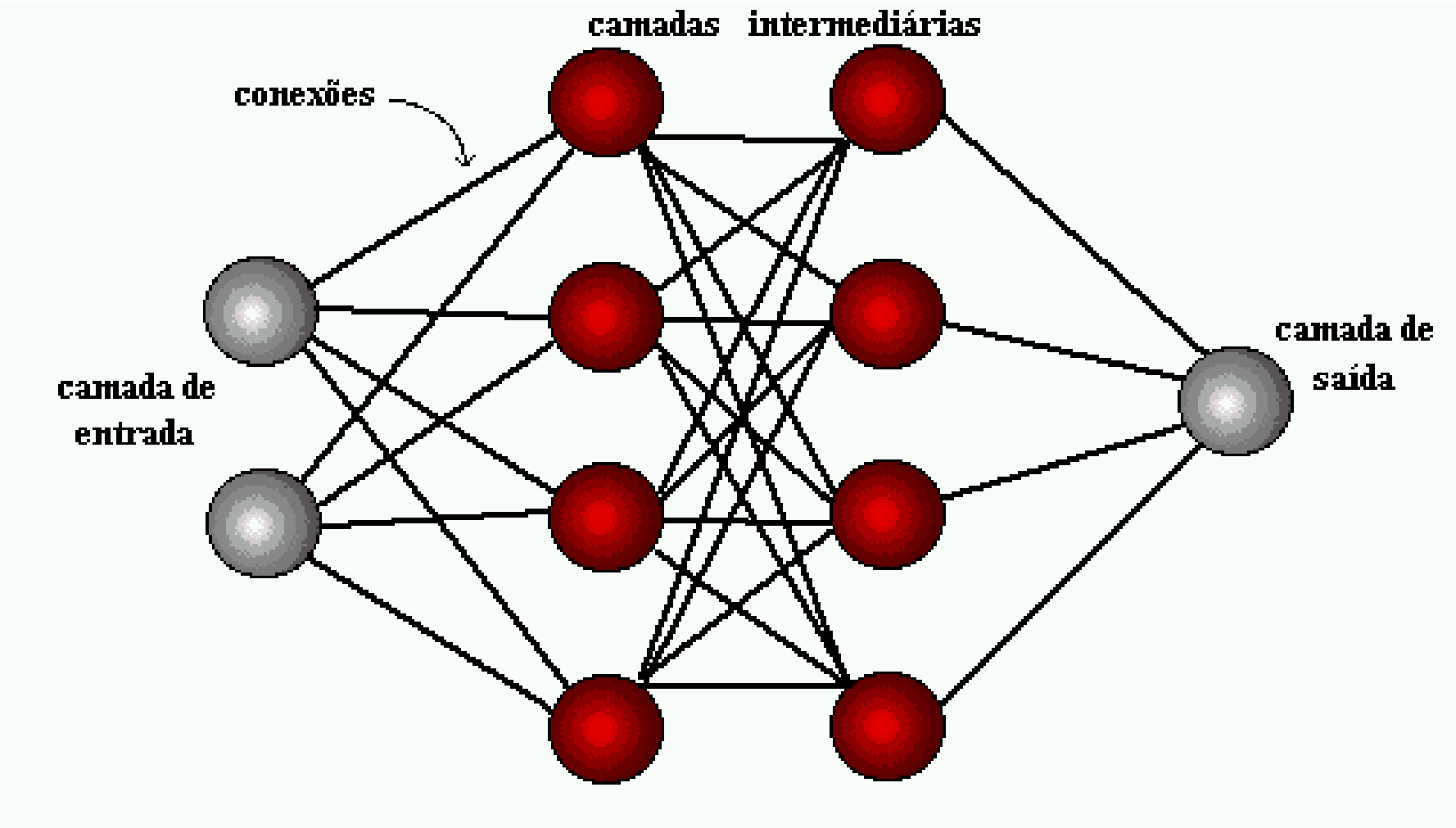
O córtex é a estrutura responsável pelas habilidades cognitivas superiores, tais como memória, raciocínio lógico, linguagem, consciência, dentre outras.



Redes Neurais Artificiais (RNA)

- Uma rede neural artificial é composta por **várias unidades de processamento (neurônios artificiais)**, cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a um determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede.

Exemplo de Topologia de uma RNA



Características

- São modelos adaptativos treináveis
- Podem representar domínios complexos (não lineares)
- São capazes de generalização diante de informação incompleta
- Robustos
- Adaptabilidade
- Capacidade de generalização
- São capazes de fazer armazenamento associativo de informações
- Processam informações espaço/temporais
- Possuem grande paralelismo, o que lhe conferem rapidez de processamento
- Tolerância a falhas
- Implementação rápida

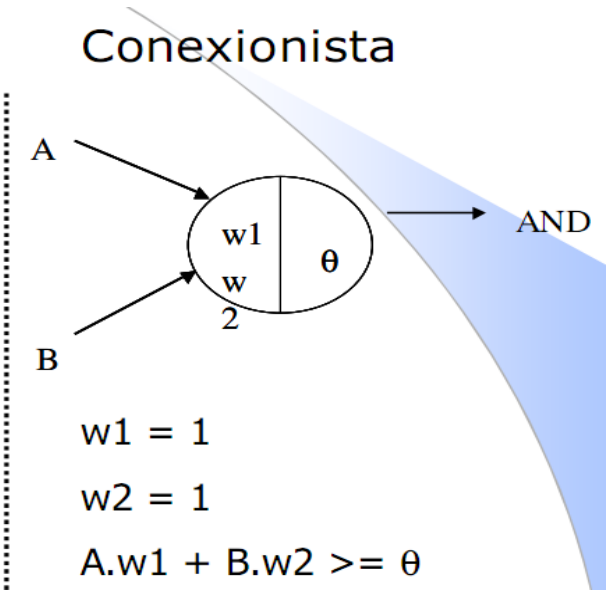
IA Simbólica vs IA Conexionista

A grande premissa do **conexionismo** para aplicações em processamento de informações e/ou inteligência artificial é o fato de que se pode analisar um problema de acordo como funcionamento do cérebro humano.

Simbólico

```
AND (A,B)
If A = 0
Then AND = 0
Else If B=0
     Then AND = 0
     Else AND = 1
```

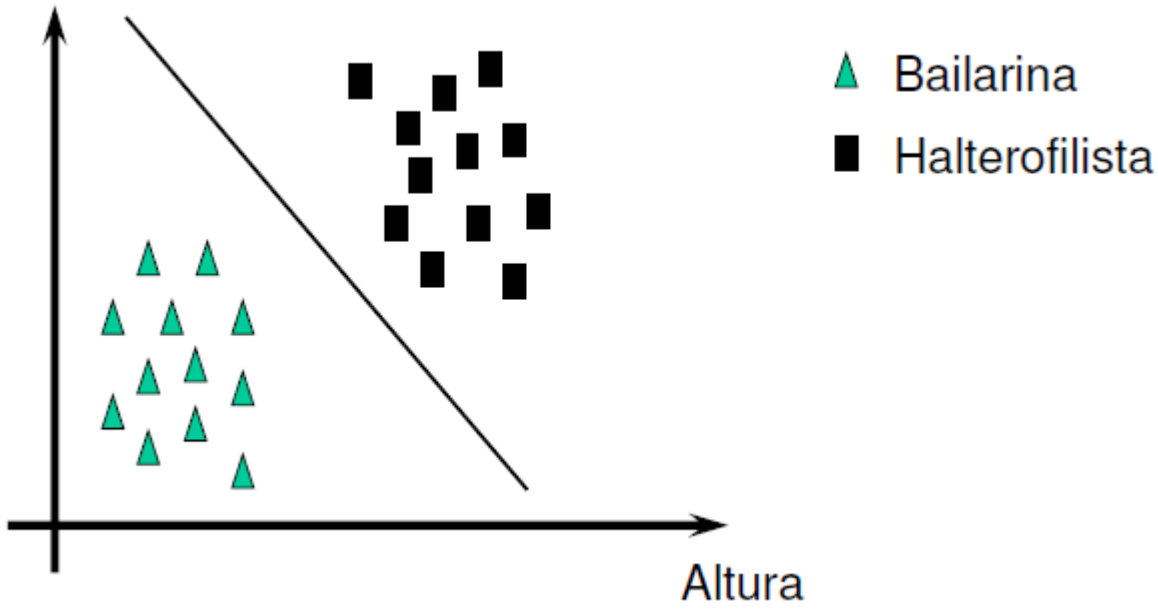
Conexionista



Aplicações de RNA

- Classificação
  - Reconhecimento de caracteres
  - Reconhecimento de imagens
  - Diagnóstico médico
  - Análise de crédito
  - Detecção de fraudes
- Categorização
  - Agrupamento de sequências de DNA
  - Mineração de dados
  - Agrupamento de clientes
- Previsão
  - Previsão do tempo
  - Previsão financeira (câmbio, bolsa...)

Reconhecimento de padrões



História

- Inter-relação entre
  - Investigação do comportamento e estrutura do sistema nervoso através de experimentação e modelagem biológica;
  - Desenvolvimento de modelos matemáticos e suas aplicações para a solução de vários problemas práticos.

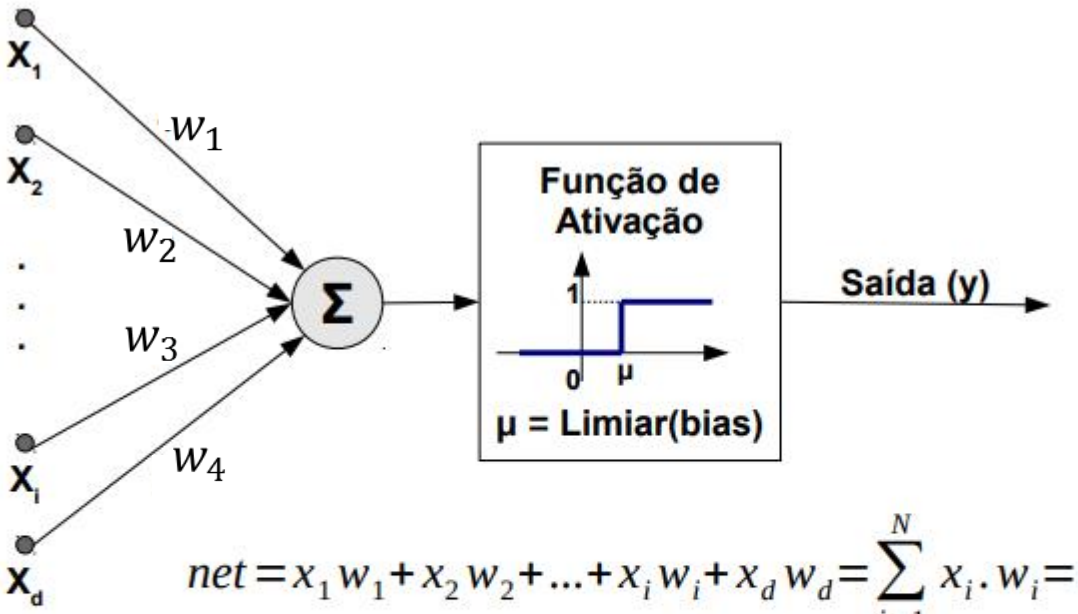
História

- (1943) McCulloch & Pitts:
  - introduziram a ideia de redes neurais como máquinas computacionais .
  - Provam, teoricamente, que qualquer função lógica pode ser implementada utilizando unidades de soma ponderada e *threshold* (limiar);
- (1949) Hebb desenvolve algoritmo para treinar RNA (aprendizado Hebbiano):
  - Se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada.

História

- (1959) Rosenblatt implementa primeira RNA, a rede *Perceptron*:
  - Ajuste iterativo de pesos;
  - Prova teorema da convergência.

Modelo McCulloch e Pitts



w_2

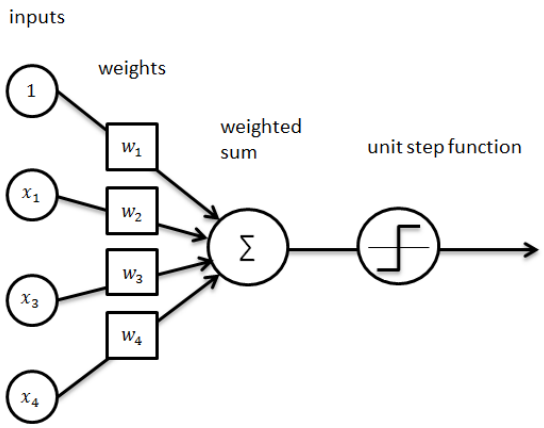
$$net = x_1 w_1 + x_2 w_2 + \ldots + x_i w_i + x_d w_d = \sum_{i=1}^{N} x_i . w_i = W \cdot X = W^T X$$

$$saída = y = f(net)$$

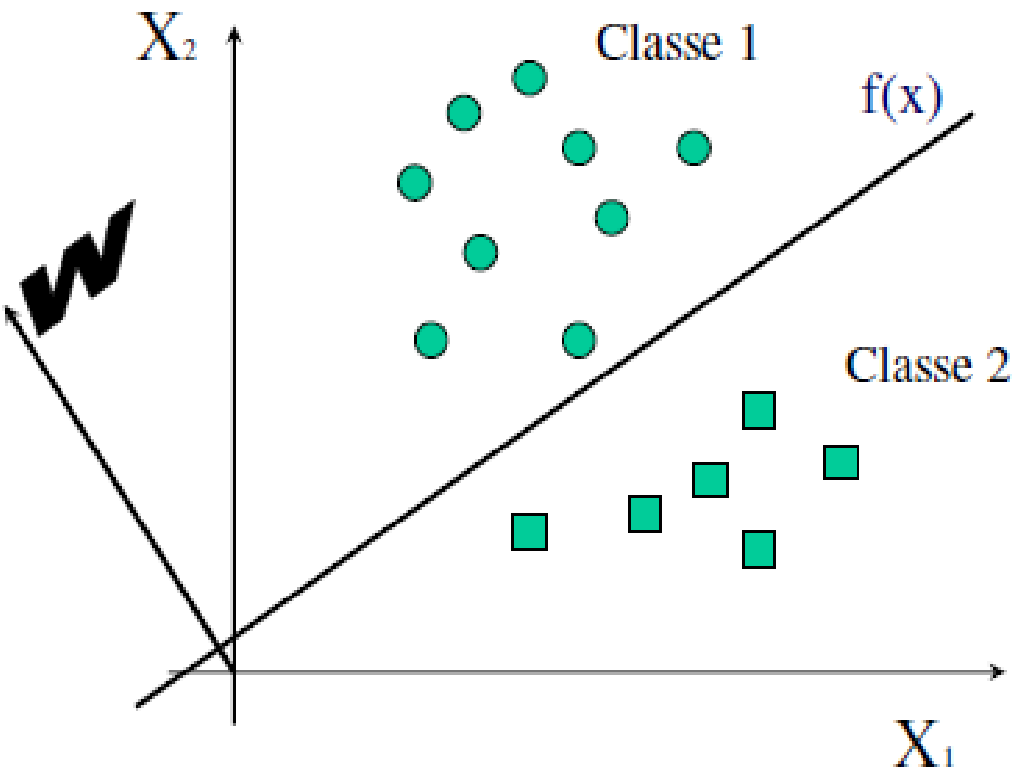
$$f(net) = \begin{cases} 1, se\ net \geq \mu \\ 0, se\ net < \mu \end{cases}$$

Perceptron com uma Camada

- Objetivo:
  - Atuar como classificador e como gerador de funções lógicas binárias
- Características
  - Aprendizado supervisionado
  - Representação binária
  - Apenas uma camada de pesos ajustáveis



Uma Visão Matemática do Perceptron



$$f(x)=\Sigma w_i.x_i - \theta$$

Considere o ponto onde $f(x) = 0$:

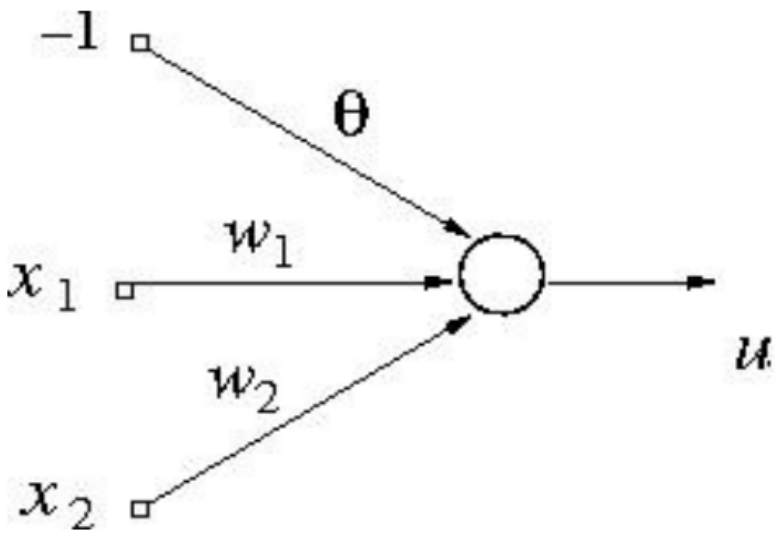
$$w_1.x_1 + w_2.x_2 - \theta = 0$$

$$\downarrow$$

$$x_2 = - w_1/w_2.x_1 + \theta/ w_2$$

$(y = m.x + c)$

Seja o neurônio artificial mostrado na figura abaixo.



x_1, x_2: entradas

w_1, w_2: pesos sinápticos

θ: limiar (*bias*)

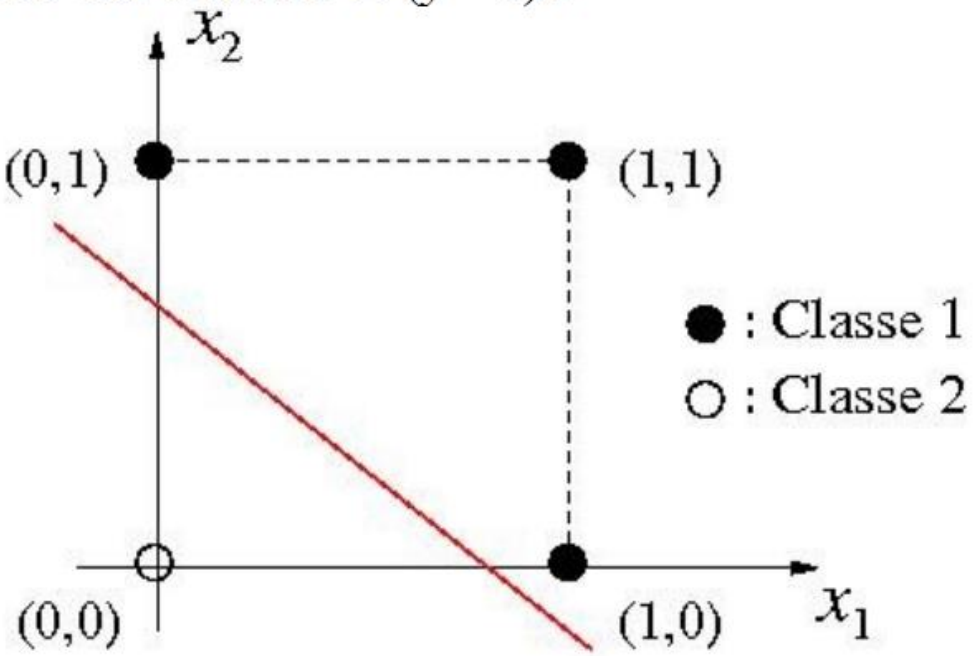
u : ativação

A ativação (u) do neurônio é dada por:

$$u = w_1 x_1 + w_2 x_2 - \theta$$

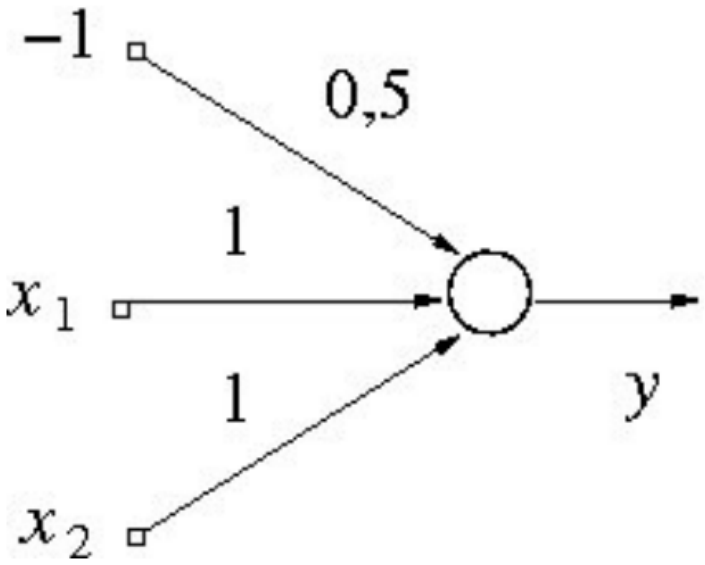
Exemplo* 1 *(cont.): É possível encontrar uma reta que separe os pontos da Classe 1 (y=1) dos da Classe 2 (y=0)?

Resposta: SIM!



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

Exemplo 2 : O seguinte neurônio implementa a porta OR.

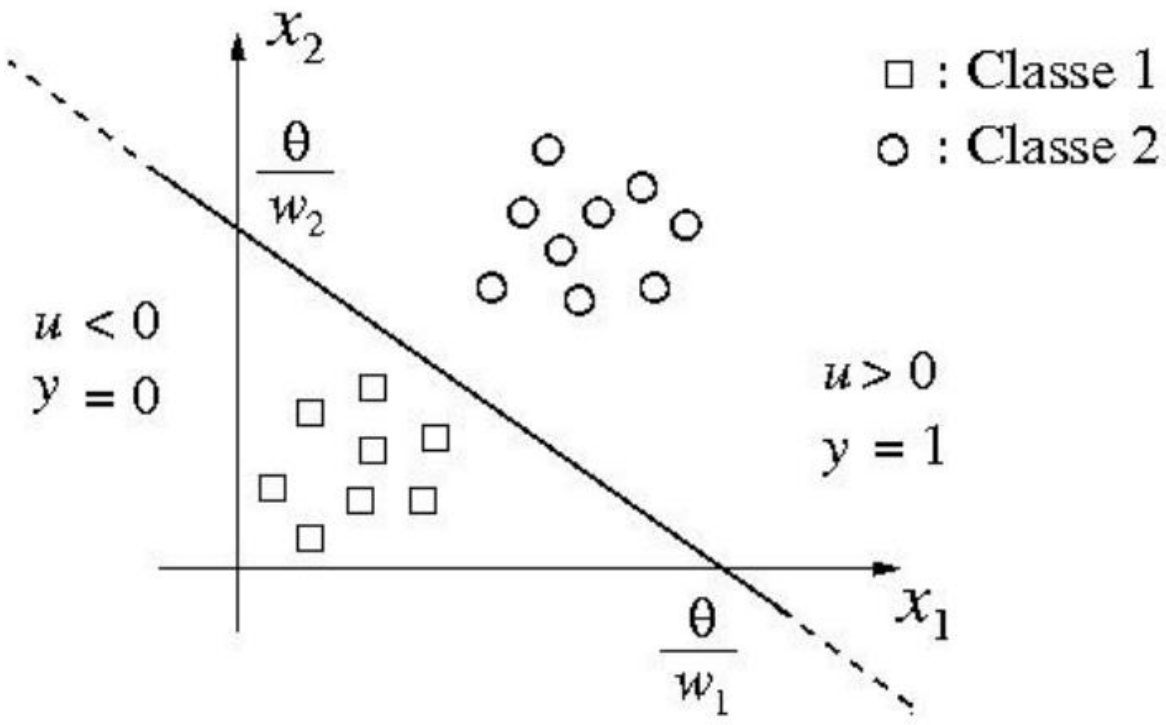


$w_1 = w_2 = 1$ e $\theta = 0{,}5$

$y = 1$, se $u \geqslant 0$.

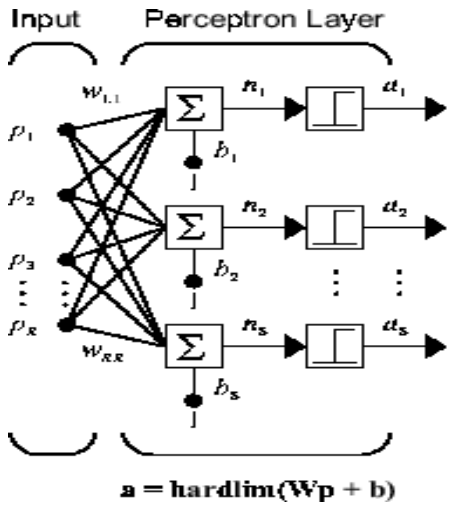
$y = 0$, se $u < 0$.

Assim, um neurônio pode ser usado para separar com eficiência duas classes que estejam bem isoladas uma da outra.



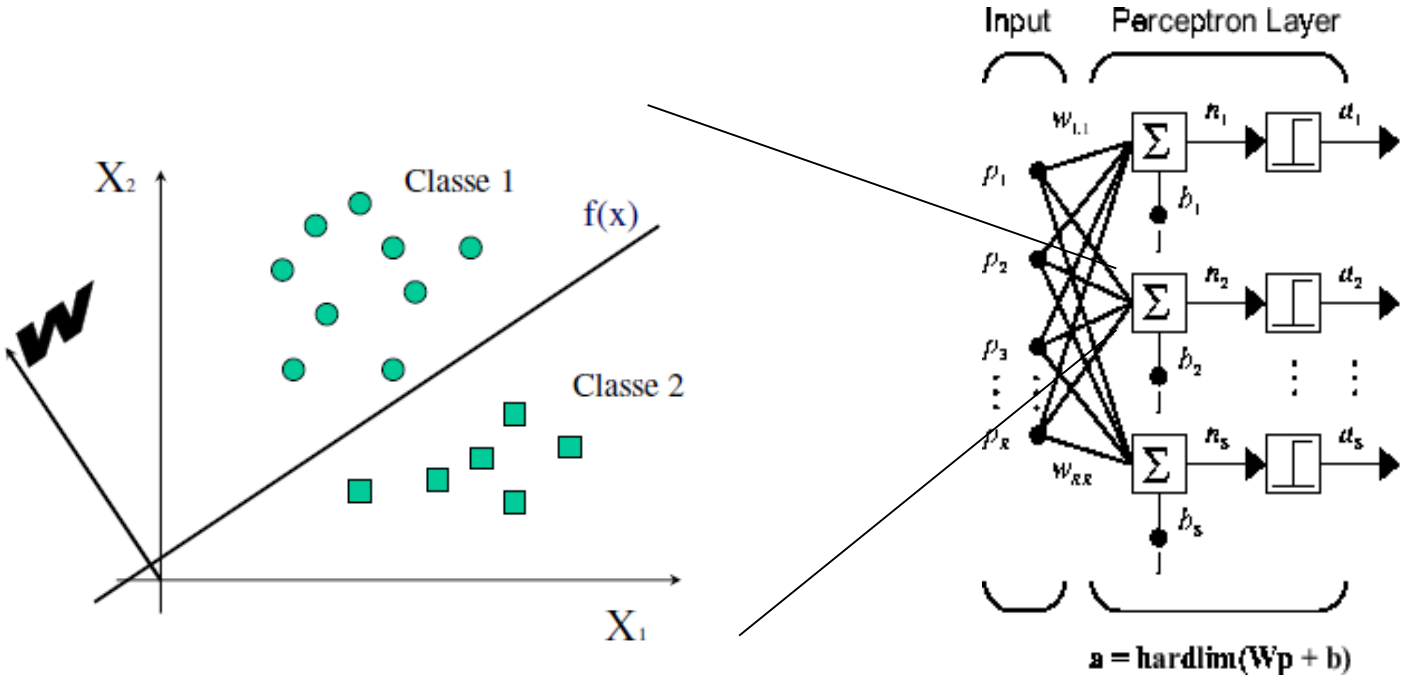
Vários perceptron com uma Camada

- Um perceptron de neurônio único pode classificar vetores de entrada em duas categorias, uma vez que a saída pode ser 0 ou 1. Um perceptron de neurônio múltiplo pode classificar entradas em ***muitas categorias***. Cada categoria é representada por um vetor de saída diferente. Uma vez que cada elemento do vetor de saída pode ser 0 ou 1, existem um total de 2^S categorias possíveis, onde *s* é o número de neurônios.



Vários perceptron com uma Camada

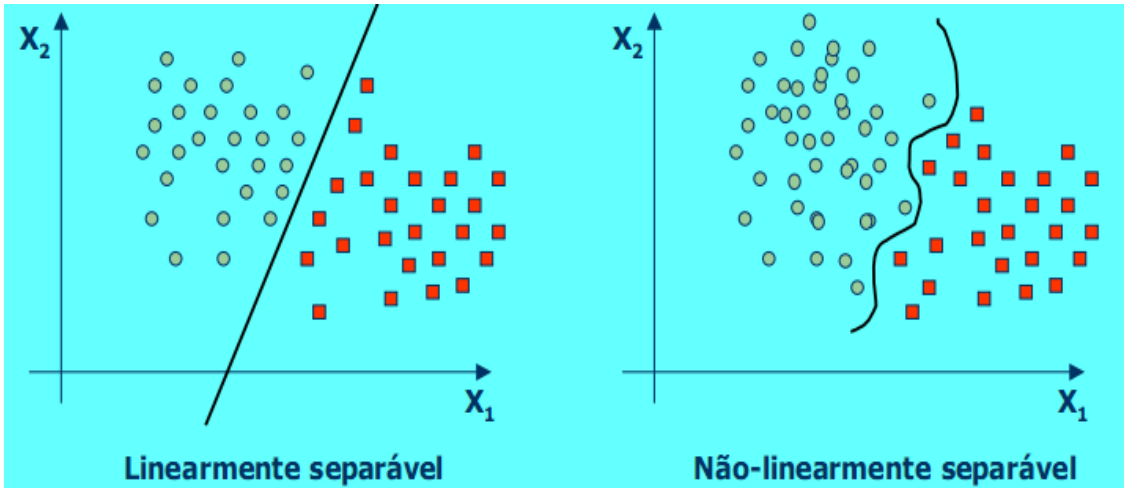
- Cada neurônio na rede divide o espaço de entrada em duas regiões



A fronteira de decisão para o neurônio *i*: $_i\mathbf{w}^T\mathbf{p} + b_i = 0.$

Características do Perceptron de uma camada

- Simples Operação
- Aprendizado nem sempre ocorre.
- As duas classes C1 e C2 devem ser **linearmente separáveis**



Função de Ativação

Nome da função	Uso/Características	Função
Linear	Usada para buffers de entrada e saída de dados. É usada também em BSB e Hopfield.	$f(s) = s$
Sinal	Utilizada em paradigmas como Perceptron	$f(s) = \begin{cases} +1 & \Rightarrow s \geqslant 0 \\ -1 & \Rightarrow s < 0 \end{cases}$
Passo	Pode ser usada no lugar da função de transferência Perceptron, ou em BAM baseada em 0	$f(s) = \begin{cases} +1 & \Rightarrow s \geqslant 0 \\ 0 & \Rightarrow s < 0 \end{cases}$
Hopfield/BAM	Usadas em redes Hopfield e BAM. Caso s=0, mantém o valor de saída anterior	$f(s) = \begin{cases} +1 & \Rightarrow s > 0 \\ -1 & \Rightarrow s < 0 \\ inateraldo & \Rightarrow s = 0 \end{cases}$

Função de Ativação

BSB ou Limiar Lógico	Linear entre os limites inferior e superior, prendendo os valores de saída ao limite caso algum destes seja excedido. Usado em BSB	$f(s) = \begin{cases} -k & \Rightarrow s \leqslant -k \\ s & \Rightarrow -k < s < +k \\ +k & \Rightarrow s \geqslant +k \end{cases}$
Linear de Limites Rígidos	Linear entre os limites 0 e 1, prendendo os valores de saída ao limite caso algum destes seja excedido. Usado em paradigmas de competição através de inibição	$f(s) = \begin{cases} 0 & \Rightarrow s \leqslant 0 \\ s & \Rightarrow 0 < s < +1 \\ +1 & \Rightarrow s \geqslant +1 \end{cases}$
Logística	Função de transferência sigmoidal usada tradicionalmente em redes feedforward de aprendizagem backpropagation. Usada também em redes Hopfield, BAM e BSB	$f(s) = \frac{1}{1+e^{-s}}$
Hiperbólica	Função usada muitas vezes no lugar da função logística por variar de -1 a 1 e não de 0 a 1 como na função logística	$f(s) = \frac{1-e^{-2s}}{1+e^{-2s}}$
Gaussiana	Saída é simétrica em torno de um centro associado, utilizadas em redes RBF (redes de funções de base radial).	$f(s) = e^{\left[\frac{-(x-x_m)^2}{2\varphi}\right]}$

Processos de Aprendizado

A propriedade mais importante das redes neurais é a habilidade de **aprender** de seu ambiente e com isso melhorar seu desempenho.

Isso é feito através de um **processo iterativo** de ajustes aplicado a seus pesos, o treinamento.

O aprendizado ocorre quando a rede neural atinge uma solução **generalizada** para uma classe de problemas.

Processos de Aprendizado

Para que o neurônio **McCulloch–Pitts** seja capaz de aprender sozinho a resolver um problema de classificação é necessário dotá-lo de uma **regra de aprendizagem.**

Uma regra de aprendizagem nada mais é do que uma equação que altera os valores dos pesos e do limiar em função dos **erros** cometidos durante a execução da tarefa de classificação.

Processos de Aprendizado

Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais. Estes algoritmos diferem entre si principalmente pelo modo **como os pesos são modificados**.

Como projetar então uma regra de aprendizagem?

Uma regra de aprendizagem pode serprojetada com base em

(*i*) Argumentos geométricos ou empíricos.

(*ii*) Critérios de otimização de função-custo.

Em geral, uma regra de aprendizagem tem a seguinte forma:

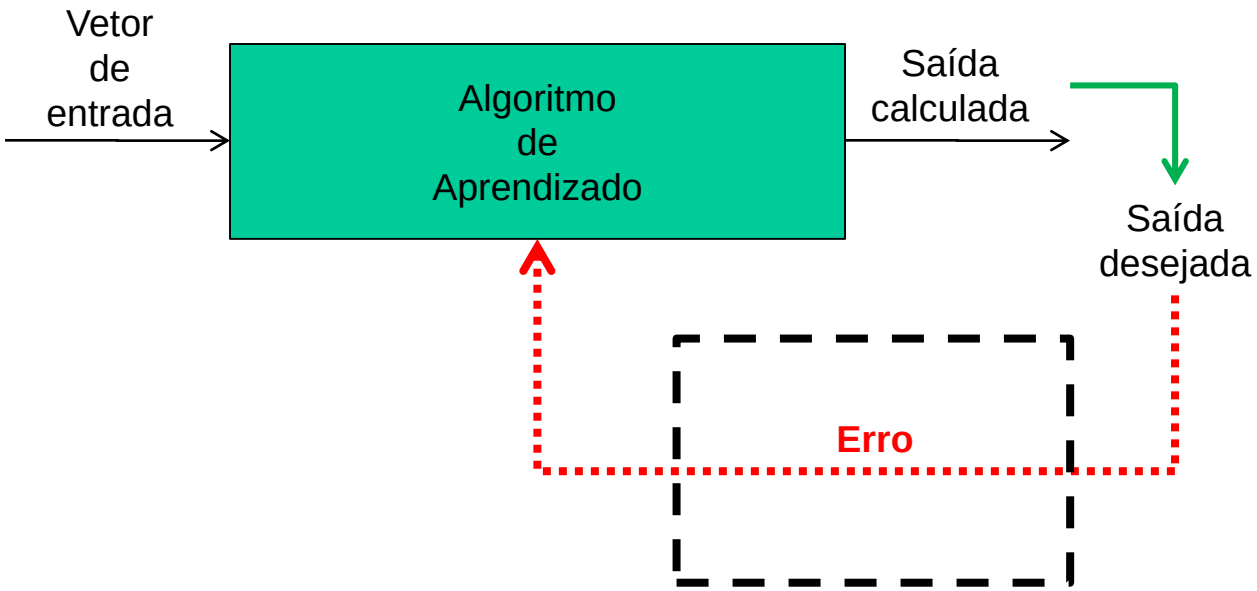
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

$\mathbf{w}(t)$ = memória (conhecimento atual).

$\Delta\mathbf{w}(t)$ = incremento na memória (informação adquirida)

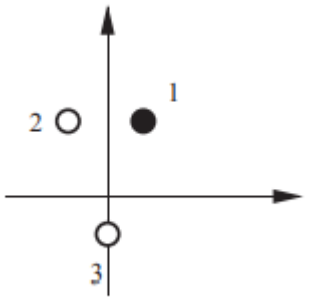
$\mathbf{w}(t+1)$ = memória modificada com acrécimo de nova informação.

Aprendizado Supervisionado



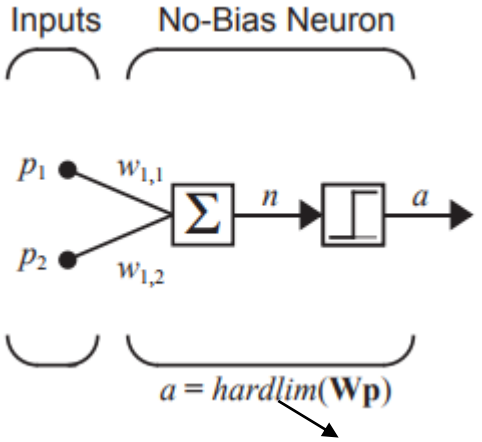
Aprendizagem do Perceptron

Exemplo: $\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}.$



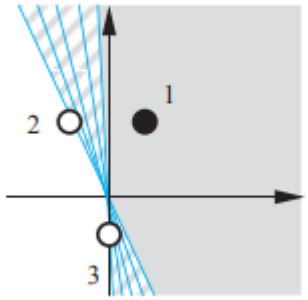
Característica da rede para o problema

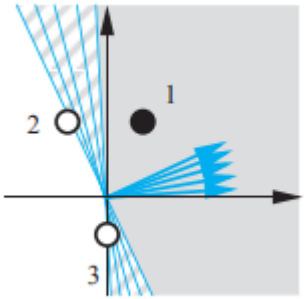
- Duas entradas, uma saída
- A príncipio vamos ignorar o bias
  - Por simplicidade
  - Porque é possível traçar uma reta que passa pelo eixo zero



Função degrau

Aprendizagem do Perceptron





Gostaríamos de uma regra de aprendizagem que encontrasse um vetor de pesos aponte em uma dessas direções

Aprendizagem do Perceptron

Aprendizado começa atribuindo algum valor inicial para os parametros da rede. Nestes caso nós estamos treinando uma rede de duas entradas e uma saída sem bias, assim nós apenas temos que inicializar seus dois pesos.

Definindo randomicamente temos:

$$_1\mathbf{w}^T = \begin{bmatrix} 1.0 & -0.8 \end{bmatrix}.$$

Aprendizagem do Perceptron

Nós agora apresentaremos os vetores de entrada para a rede. Nós começamos com p1

Função degrau

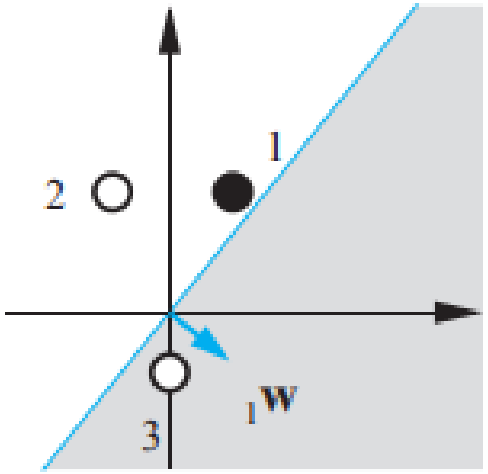
$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix}1.0 & -0.8\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)$$

$$a = hardlim(-0.6) = 0\ .$$

A rede não retornou o valor correto. A saída da rede é **zero**, enquanto a resposta correta é **um.**

Aprendizagem do Perceptron

Nós precisamos alterar o vetor de pesos a fim de apontar mais em direção a p1, para que no futuro isto tem uma chance melhor de classificar isto corretamente.

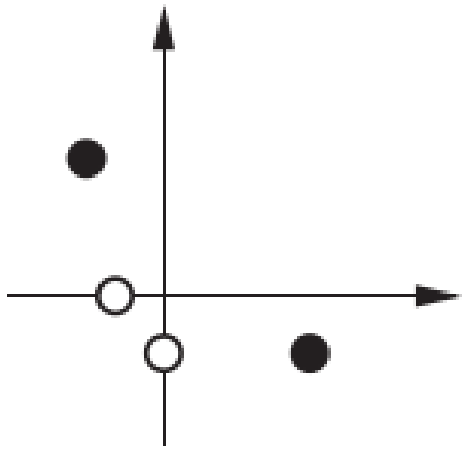


Uma solução rápida de aprendizagem: $_1\boldsymbol{w} = \mathbf{p1}$.

Aprendizagem do Perceptron

Contudo esta idéia de aprendizagem pode não conseguir aprender.

Os pesos da rede simplesmente vão oscilar para atrás e pra frente.



Aprendizagem do Perceptron

Uma outra possibilidade seria adicionar p1 a ${}_1\boldsymbol{W}$.

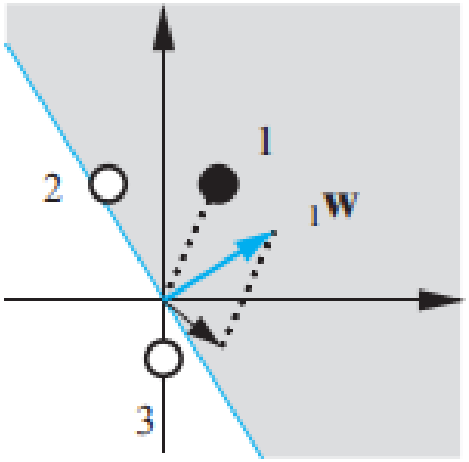
Adicionando p1 a ${}_1\boldsymbol{W}$ faria ${}_1\boldsymbol{W}$ apontar mais nada direção de p1. Apresentações repetidas de p1 levaria a direção de ${}_1\boldsymbol{W}$ a assintoticamente a apontar para a direção de p1. Esta regra pode ser estabelecida como:

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}.$$

Aprendizagem do Perceptron

Aplicando esta regra para o nosso problema teste para definir novos valores de ${}_1\boldsymbol{w}$:

$$ {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}. $$



Aprendizagem do Perceptron

O próximo vetor de entrada é p2. Quando é apresentado a rede nós encontramos

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_2) = hardlim\left(\begin{bmatrix} 2.0 & 1.2 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} \right)$$
$$= hardlim(0.4) = 1 .$$

A resposta correta associado a entrada p2 é 0 e a saída da rede foi 1. Um vetor da classe 0 foi mal classificado como 1.

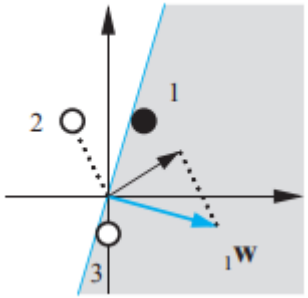
Aprendizagem do Perceptron

Uma vez que nós gostaríamos de mover o vetor de peso para longe da entrada nós podemos simplesmente mudar a adição na Equação anterior de atualização para subtração:

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}.$$

Dessa forma temos:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix},$$



Aprendizagem do Perceptron

Nós agora apresentamos o terceiro vetor p3

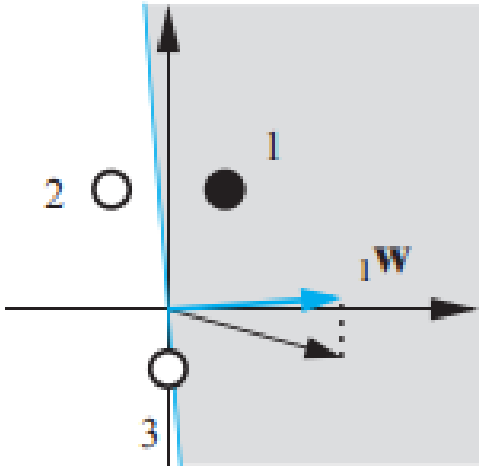
$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix} 3.0 & -0.8 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right)$$
$$= hardlim(0.8) = 1 .$$

O vetor atual w resulta uma fronteira de decisão que classifica erradamente p3. Isto é uma situação para o qual nós já temos uma regra. Assim atualizando w1 novamente temos:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}.$$

Aprendizagem do Perceptron

Resultado final. O perceptron conseguiu aprender a classificar estes dados.



Ainda temos uma regra final que não foi mencionado nos exemplos:

$$\text{If } t = a, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}.$$

Aprendizagem do Perceptron

Resumo das regras de aprendizagem utilizadas:

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}.$$

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}.$$

$$\text{If } t = a, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}.$$

Regra de Aprendizagem do Perceptron

$$\mathbf{w}(t+1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

A fim de tornar o processo de ajuste do vetor **w** mais estável, é comum introduzir na equação anterior um fator η, chamado de passo de aprendizagem:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta e(t)\mathbf{x}(t)$$

Em que $0 < \eta << 1$.

$$e_i = (d_i - y_i)$$

Passos para o treinamento

Deste modo, tem-se o seguinte esquema de treinamento:

- Iniciar todas as conexões com pesos aleatórios;
- Repita até que o erro seja satisfatoriamente pequeno
- Para cada par de treinamento faça:
- Calcular a Resposta Obtida
- Se o erro não for satisfatoriamente pequeno então
- Atualizar os pesos: peso novo := peso anterior + taxa de aprendizado

Onde:

1. O par de treinamento corresponde ao padrão de entrada e a sua respectiva resposta desejada;

2. O erro é definido como: Resposta Desejada - Resposta Obtida

3. A taxa de aprendizado é uma constante positiva, que corresponde à velocidade do aprendizado

As respostas geradas pelas unidades são calculadas através de uma função de ativação.

Algoritmo de treinamento

1) Iniciar todas as conexões com $w_{ij} = 0$;

2) Repita

Para cada par de treinamento (X, d)

Calcular a saída y

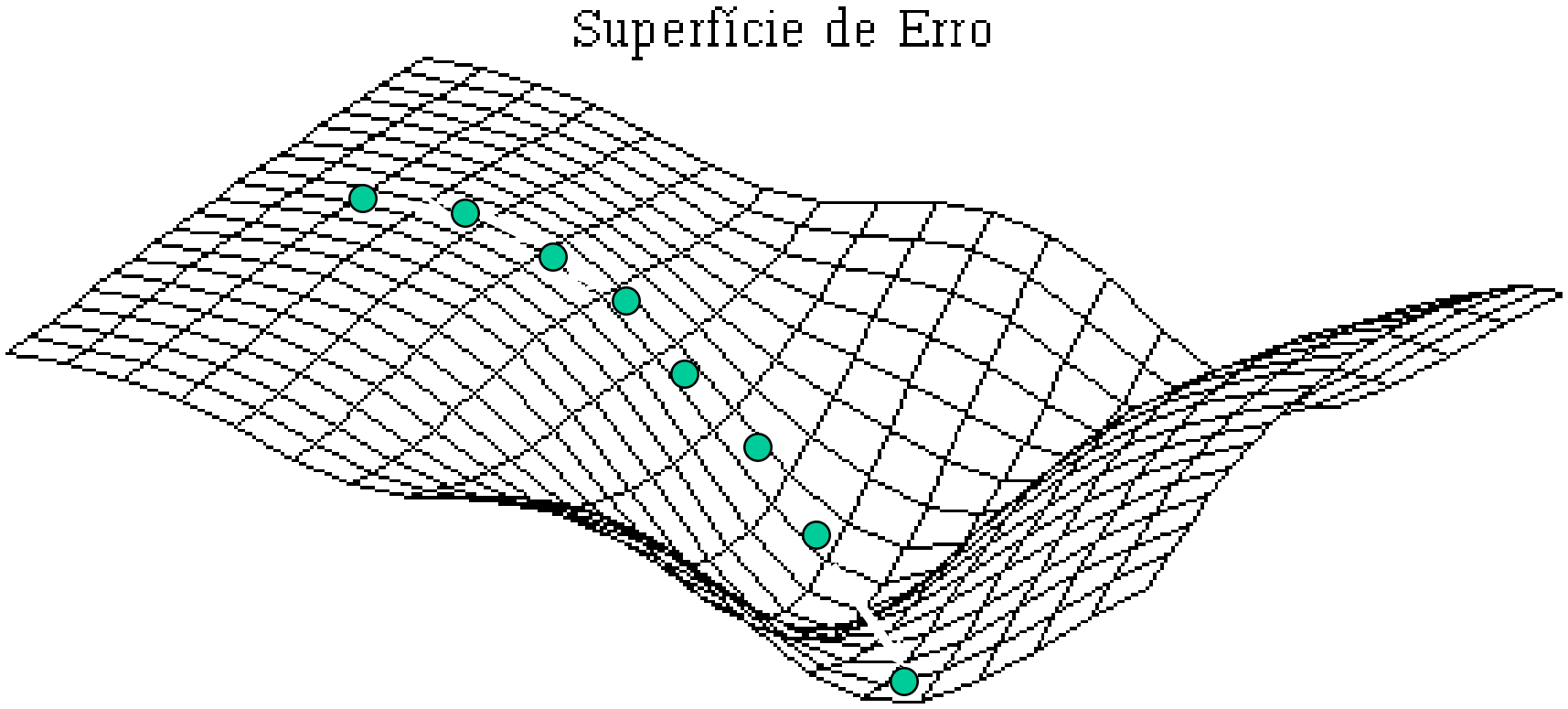
Se ($d \neq y$)

Então

Atualizar pesos dos neurônios

Até o erro ser aceitável

Treinamento



Algoritmo de teste

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

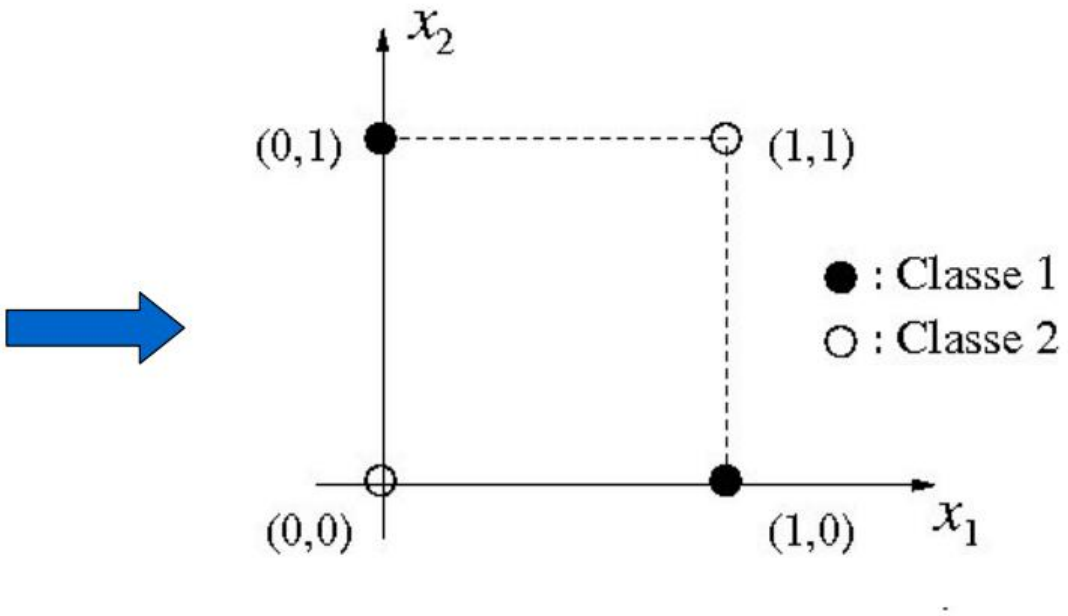
Limitações do Perceptron

Exemplo: É possível implementar a porta XOR com 1 neurônio?

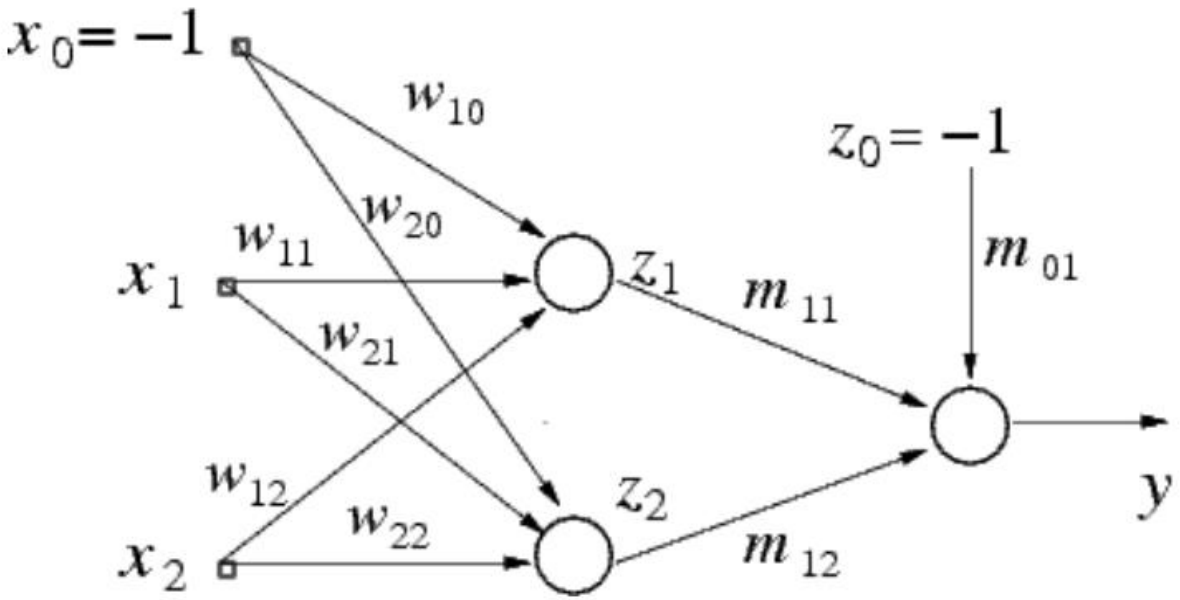
Representação do Problema (Função XOR)

Porta XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



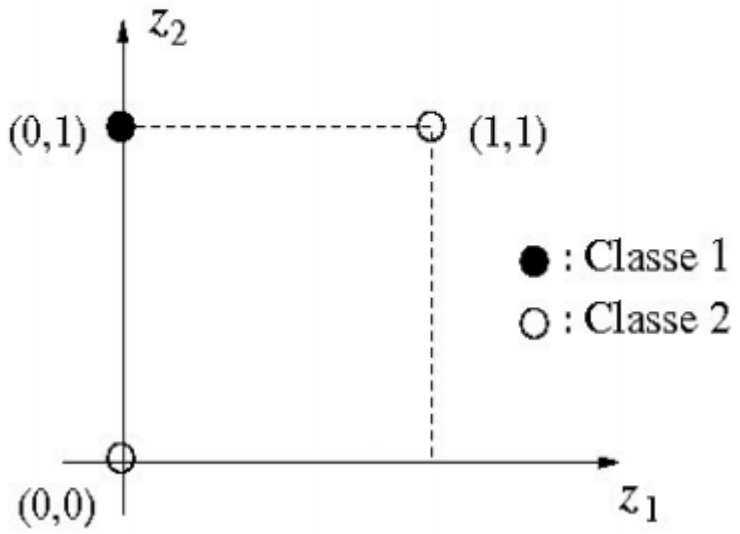
***Exemplo* (cont.)** : <u>São necessários pelo menos TRÊS neurônios!!</u>



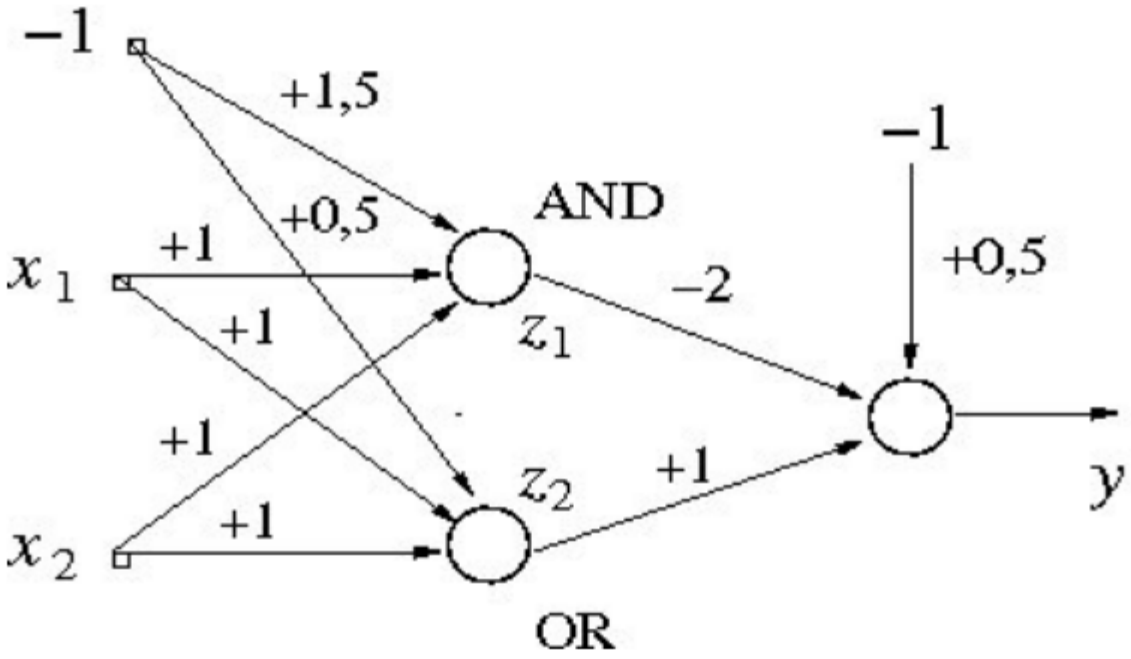
***Exemplo* (cont.)** : Assim, devemos projetar um neurônio que implemente a seguinte função lógica no espaço (z_1, z_2).

Porta logica

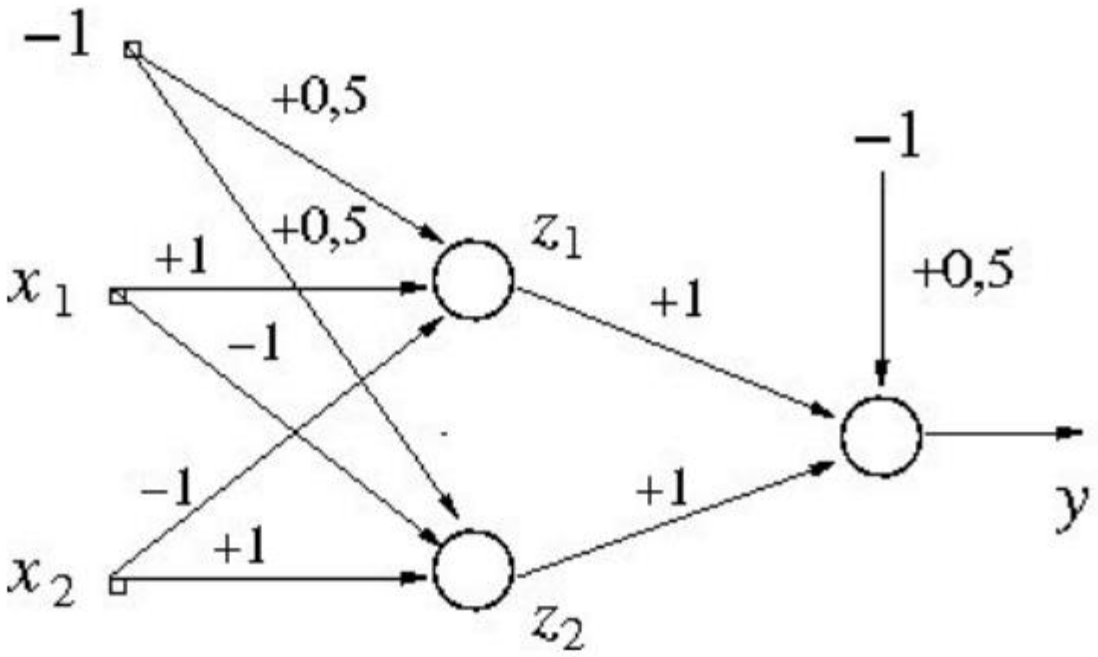
z_1	z_2	y
0	0	0
0	1	1
1	1	0



***Exemplo* (cont.)** : Colocando os dois primeiros neurônios em uma camada e o terceiro neurônio na camada seguinte (subseqüente), chega-se à seguinte rede multicamadas que implementa a porta XOR.



***Exemplo* (cont.)** : Uma outra possível rede multicamadas que também implementa a porta lógica XOR é dada abaixo.



Redes de mais de uma camada é chamada de Multi Layer Perceptron (MLP)

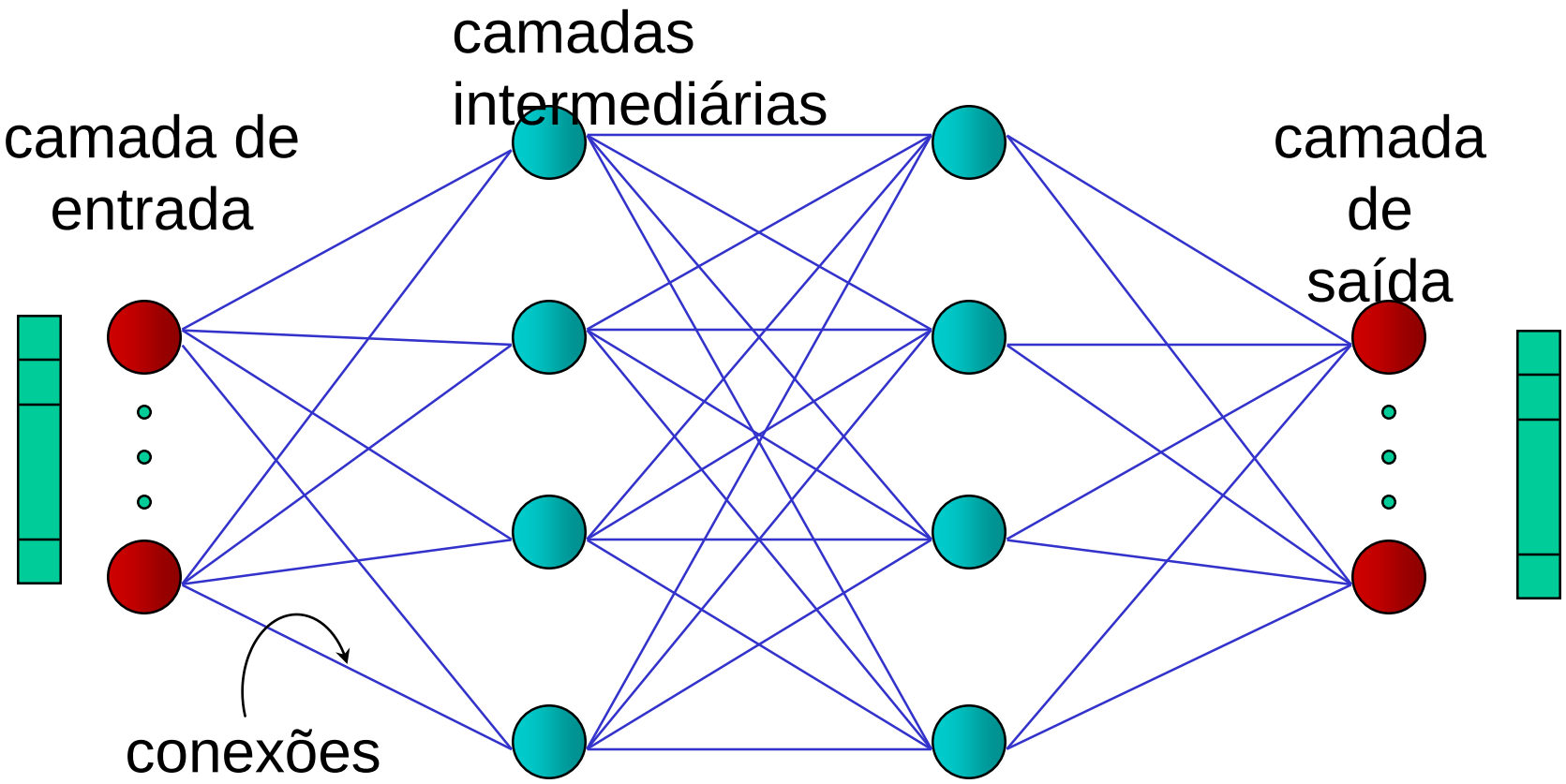
MLP - Introdução

Redes de uma camada resolvem apenas problemas **linearmente separáveis**

Solução: utilizar mais de uma camada

- Camada 1: uma rede Perceptron para cada grupo de entradas linearmente separáveis
- Camada 2: uma rede combina as saídas das redes da primeira camada, produzindo a classificação final

Topologia de uma MLP genétrica



Câmadas do Multilayer Percetron

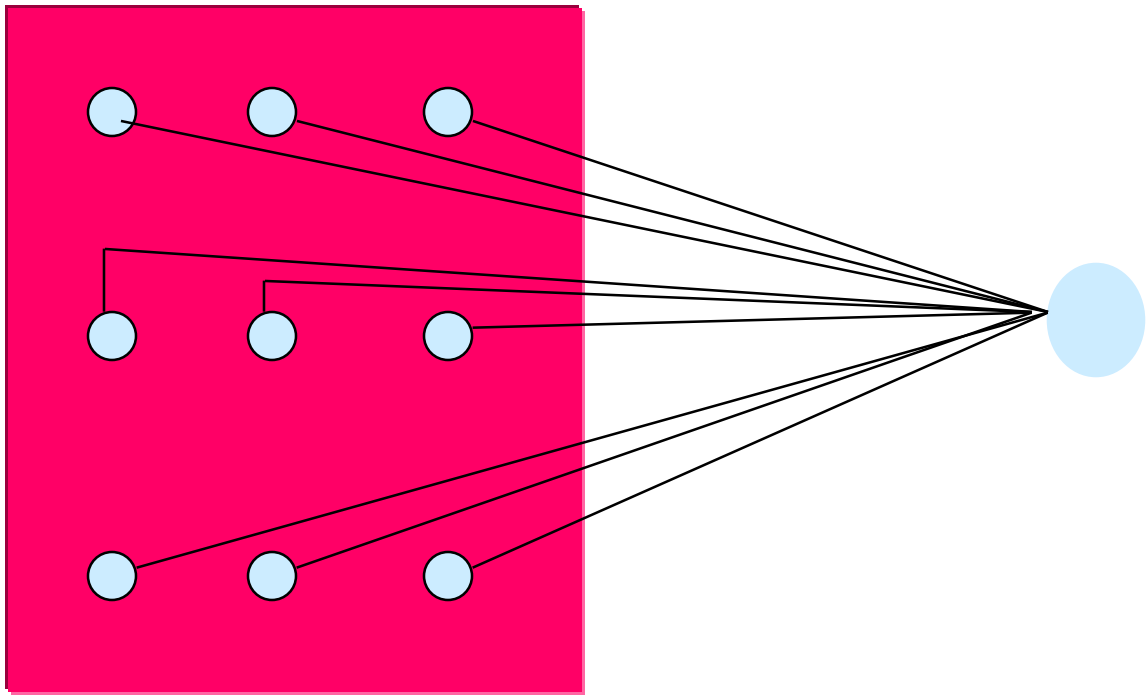
Usualmente as camadas são classificadas em três grupos:

- **Camada de Entrada**: onde os padrões são apresentados à rede;
- **Camadas Intermediárias ou Escondidas**: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características; **Essa etapa também é responsável pelo processamento não-linear da informação de entrada de modo a facilitar a resolução do problema para os neurônios da camada de saída.**
- **Camada de Saída**: onde o resultado final é concluído e apresentado.

Conexões do MLP

- Completamente conecta
- Parcialmente conectada
- Localmente conectada

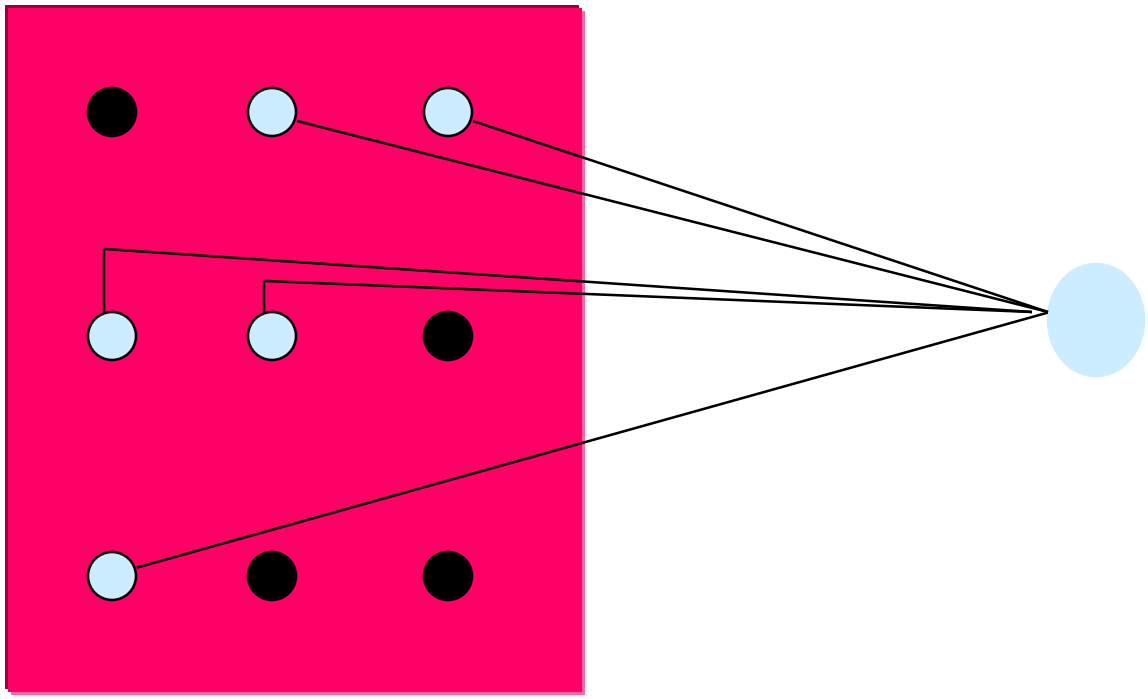
Completamente conectada



Camada i

Camada i +1

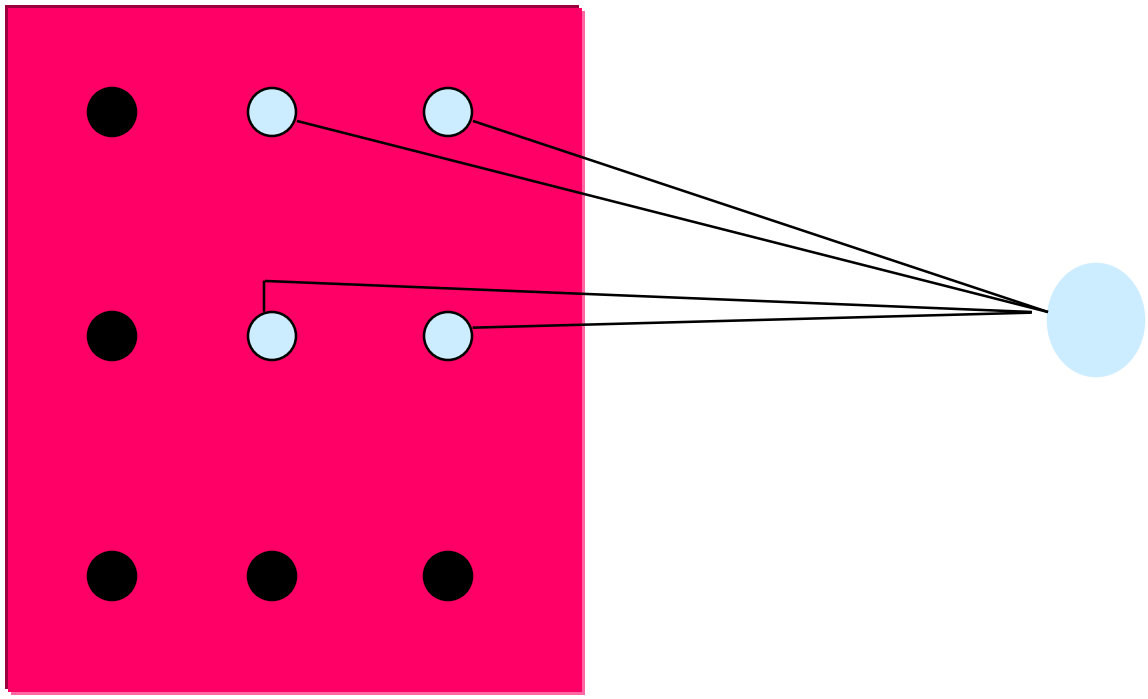
Parcialmente conectada



Camada i

Camada i +1

Localmente conectada



Camada i

Camada i +1

Arranjo das conexões

Redes feedforward

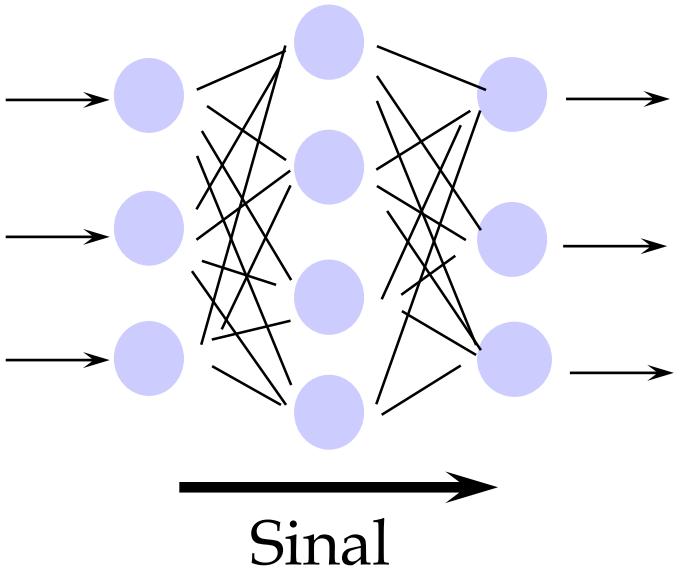
- Não existem loops de conexões

Redes recorrentes

- Conexões apresentam loops
- Mais utilizadas em sistemas dinâmicos

Redes feedforward

Sinais seguem em uma única direção

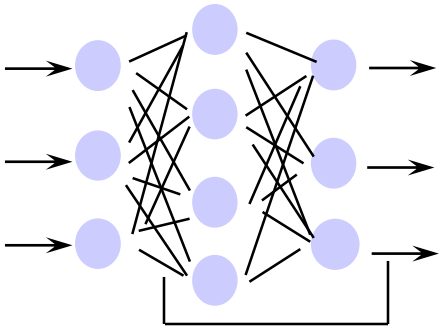


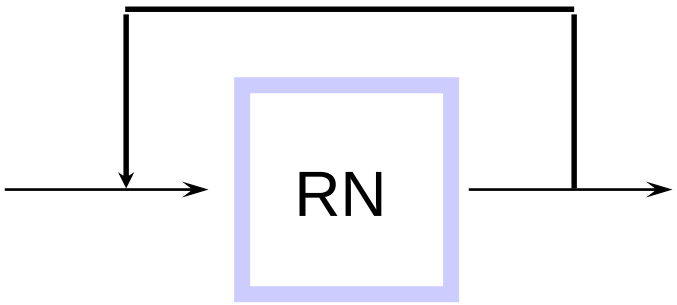
OU

- Tipo mais comum

Redes recorrentes

Possuem conexões ligando saída da rede a sua entrada



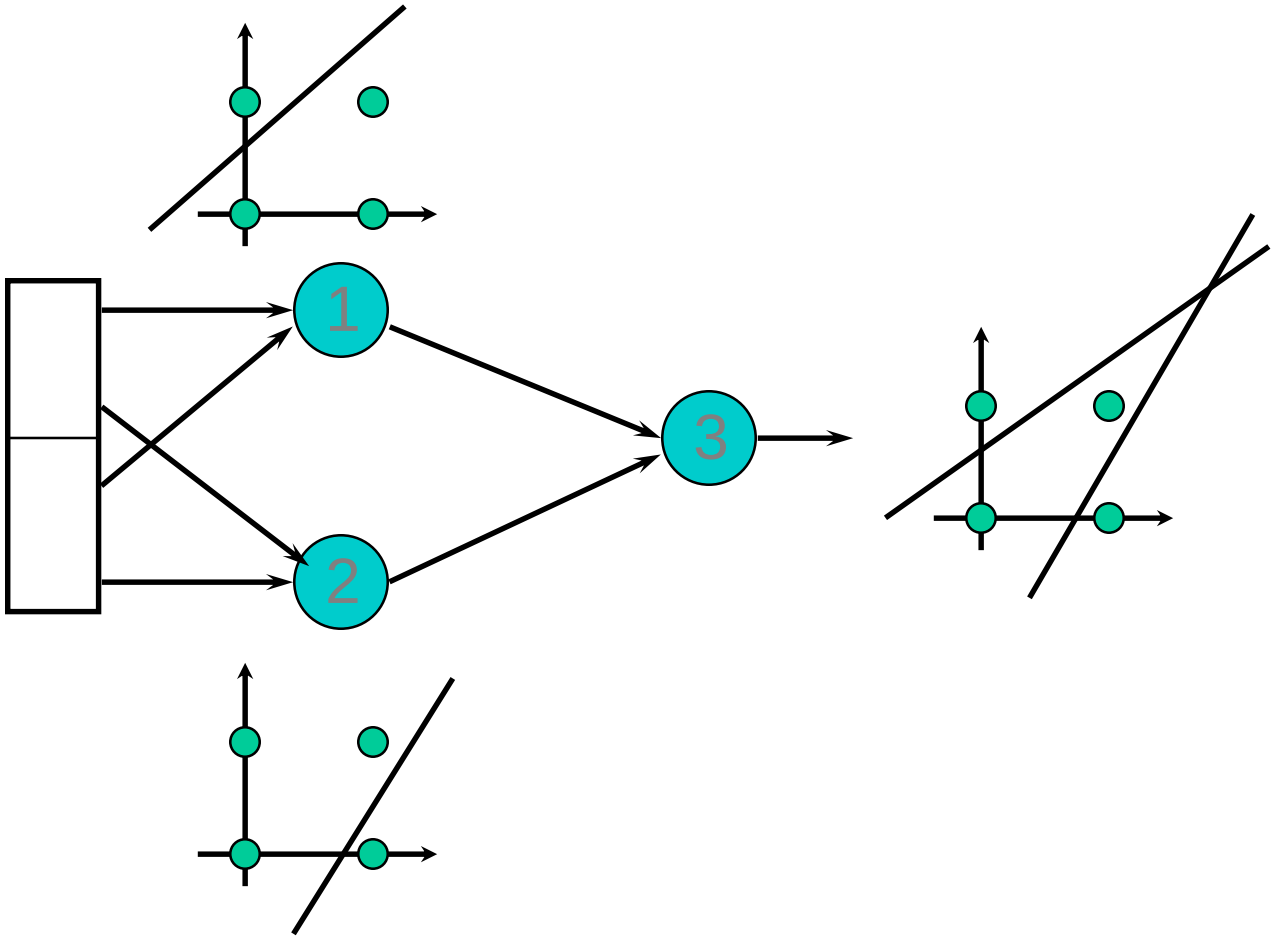


- Podem lembrar entradas passadas e, conseqüentemente, processar seqüência de informações (no tempo ou espaço)

Aplicações com redes recorrentes

- Robot control[6]
- Time series prediction[7]
- Speech recognition[8][9]
- Rhythm learning[10]
- Music composition[11]
- Grammar learning[12][13][14]
- Handwriting recognition[15][16]
- Human action recognition[17]
- Protein Homology Detection[18]

MLP



MLP

- Treinamento da rede
  - Treinar cada rede indepentemente
    - Saber como dividir o problema em sub-problemas
    - Nem sempre é possível
  - Treinar a rede toda
    - Qual o erro dos neurônios da camada intermediária?
    - Função threshold leva ao problema de atribuição de crédito
    - Usar função de ativação linear?

MLP

- Função de ativação linear
  - Cada camada computa uma função linear
    - Composição de funções lineares é uma função linear
    - Sempre vai existir uma rede com uma camada equivalente uma rede multicamadas com funções de ativação lineares

MLP

Função de ativação para redes multicamadas

- Não deve ser linear
- Deve informar os erros para as camadas inferiores da rede
  - Função sigmóide
  - Função tangente hiperbólica

Funções de ativação

Funções de ativação mais comuns

$a(t+1) = u(t)$ (linear)

$$a(t+1) = \begin{cases} 1, \text{ se } u(t) \geq \theta \\ 0, \text{ se } u(t) < \theta \end{cases}$$ (threshold ou limiar)

$a(t+1) = 1/(1 + e^{-\lambda u(t)})$ (sigmoid logística)

$$a(t+1) = \frac{(1 - e^{-\lambda u(t)})}{(1 + e^{-\lambda u(t)})}$$ (tangente hiperbólica)

Treinamento

- A rede neural supervisonada MLP utiliza métodos derivados do gradiente no ajustes de seus pesos por retropropagação.

  – Esta rede consiste de uma camada de entrada, uma ou mais camadas escondidas e uma camada de saída. Um sinal de entrada é propagado, de camada em camada, da entrada para a saída.

  – MLP é treinada com um algoritmo de retropropagação do erro.

- Estágios da aprendizagem por retropropagação do erro:

  – **Passo para frente**: Estímulo aplicado à entrada é propagado para frente até produzir resposta da rede.

  – **Passo para trás**: O sinal de erro da saída é propagado da saída para a entrada para ajuste dos pesos sinápticos.

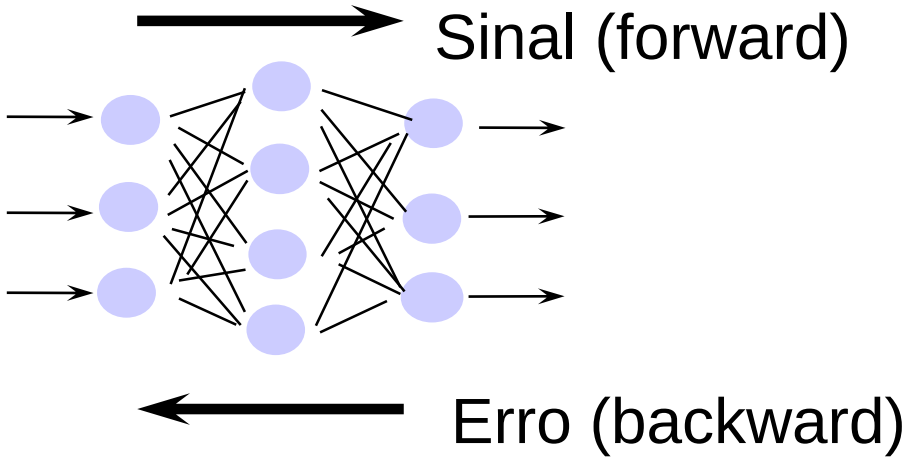
Treinamento - Backpropagation

Rede é treinada com pares entrada-saída

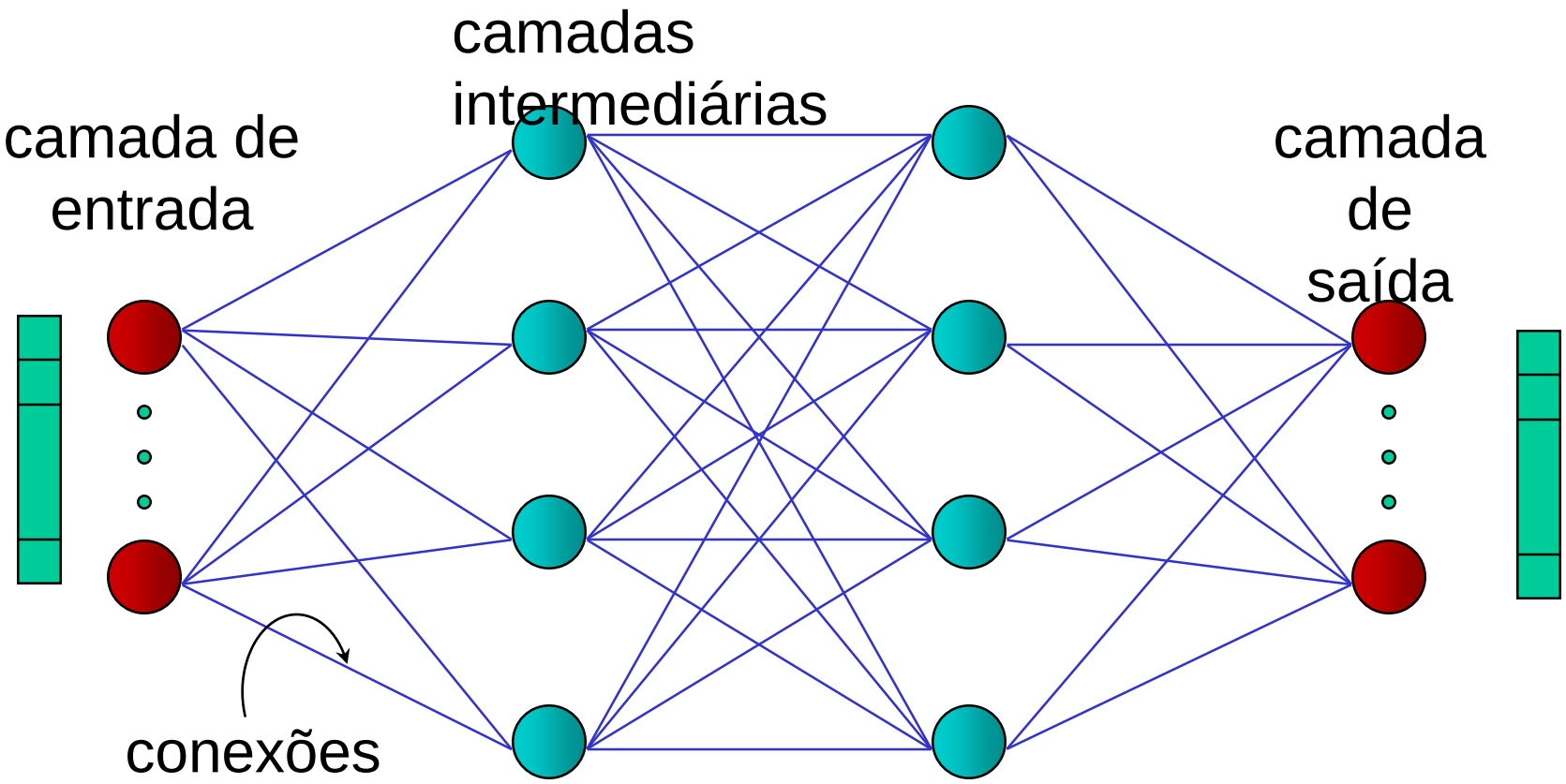
Cada entrada de treinamento está associada a uma saída desejada

Treinamento em duas fases, cada uma percorrendo a rede em um sentido

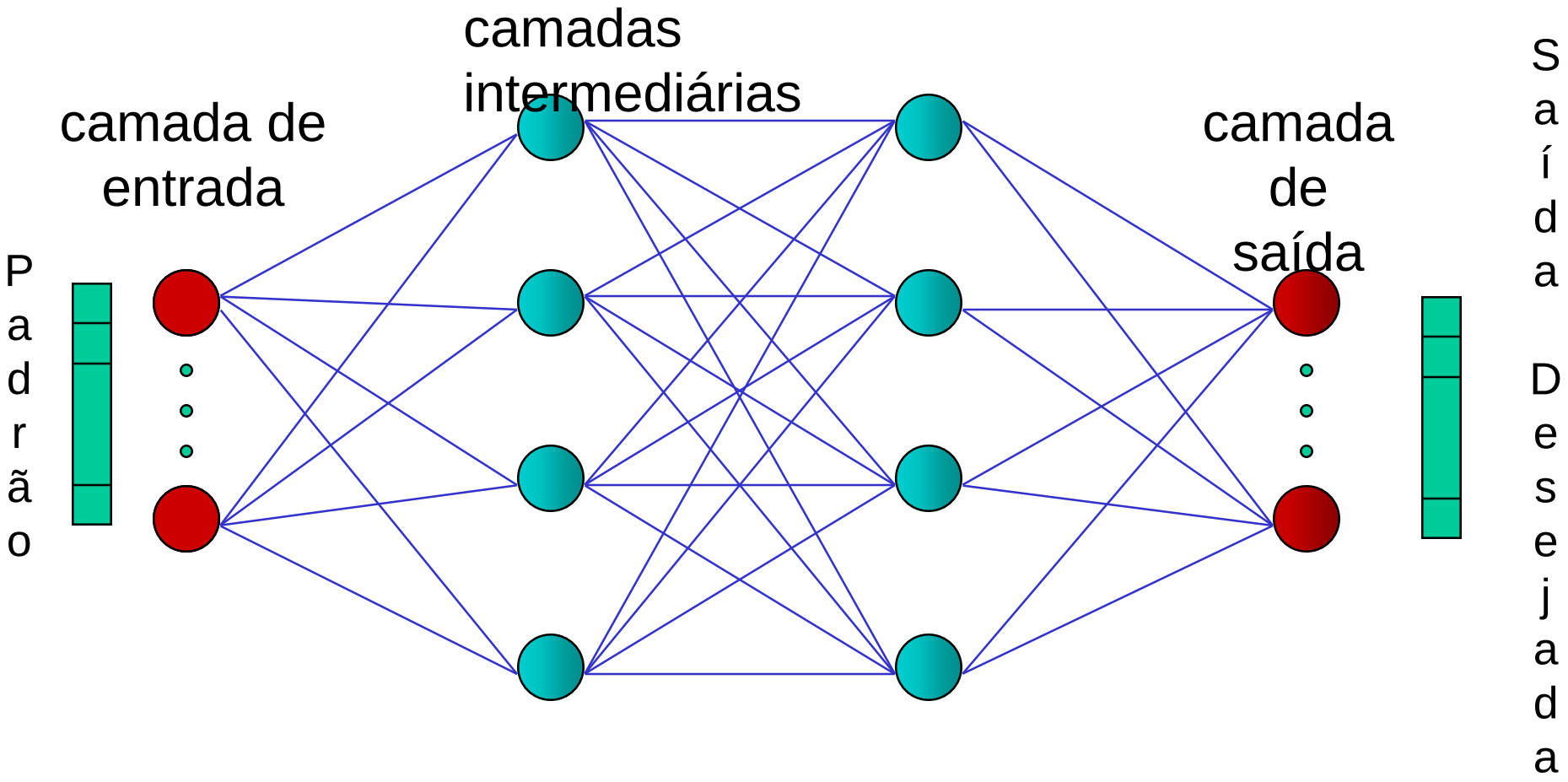
- Fase forward
- Fase backward



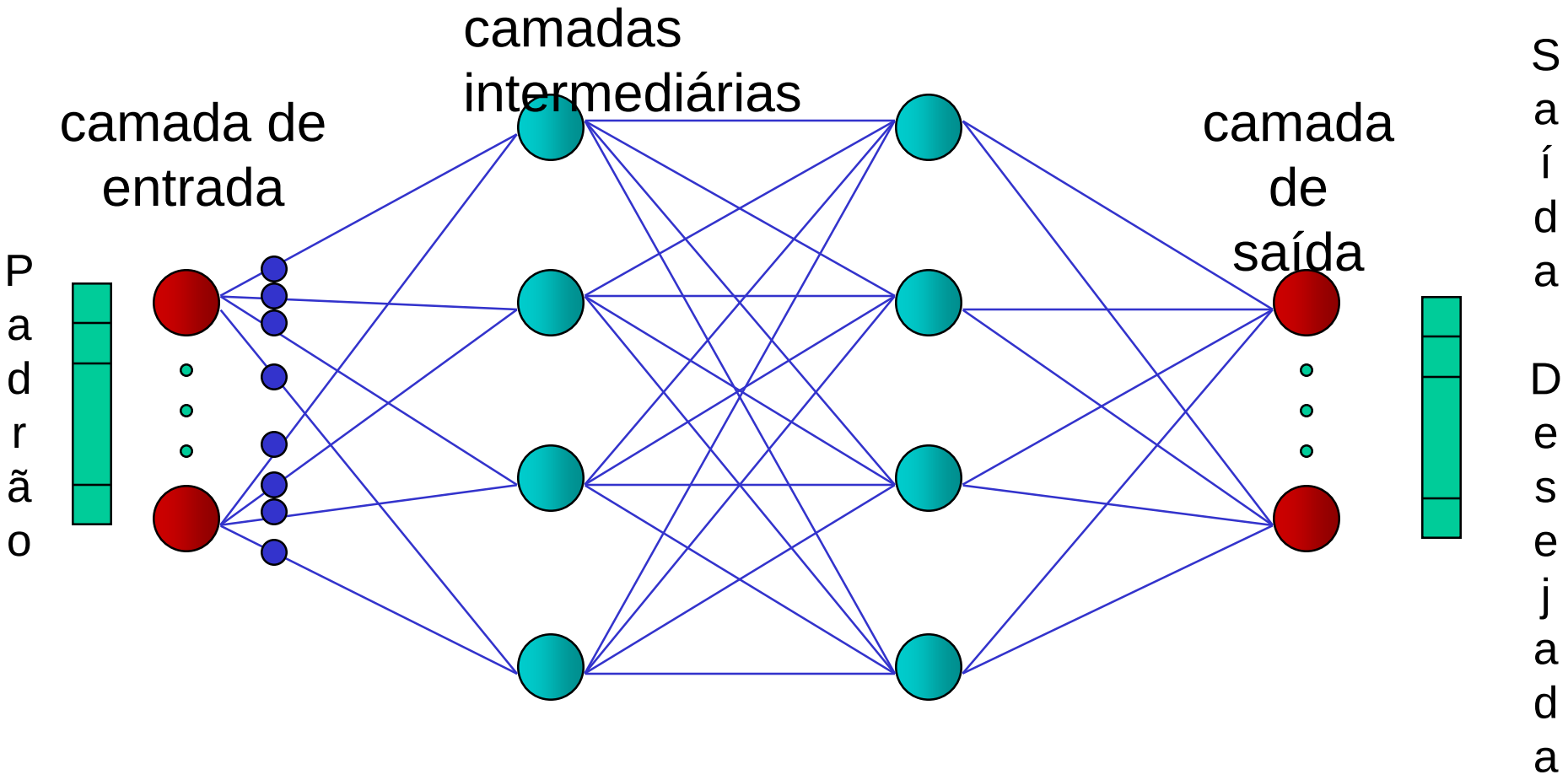
Rede MLP


camadas
intermediárias
camada de
entrada
camada
de
saída
conexões

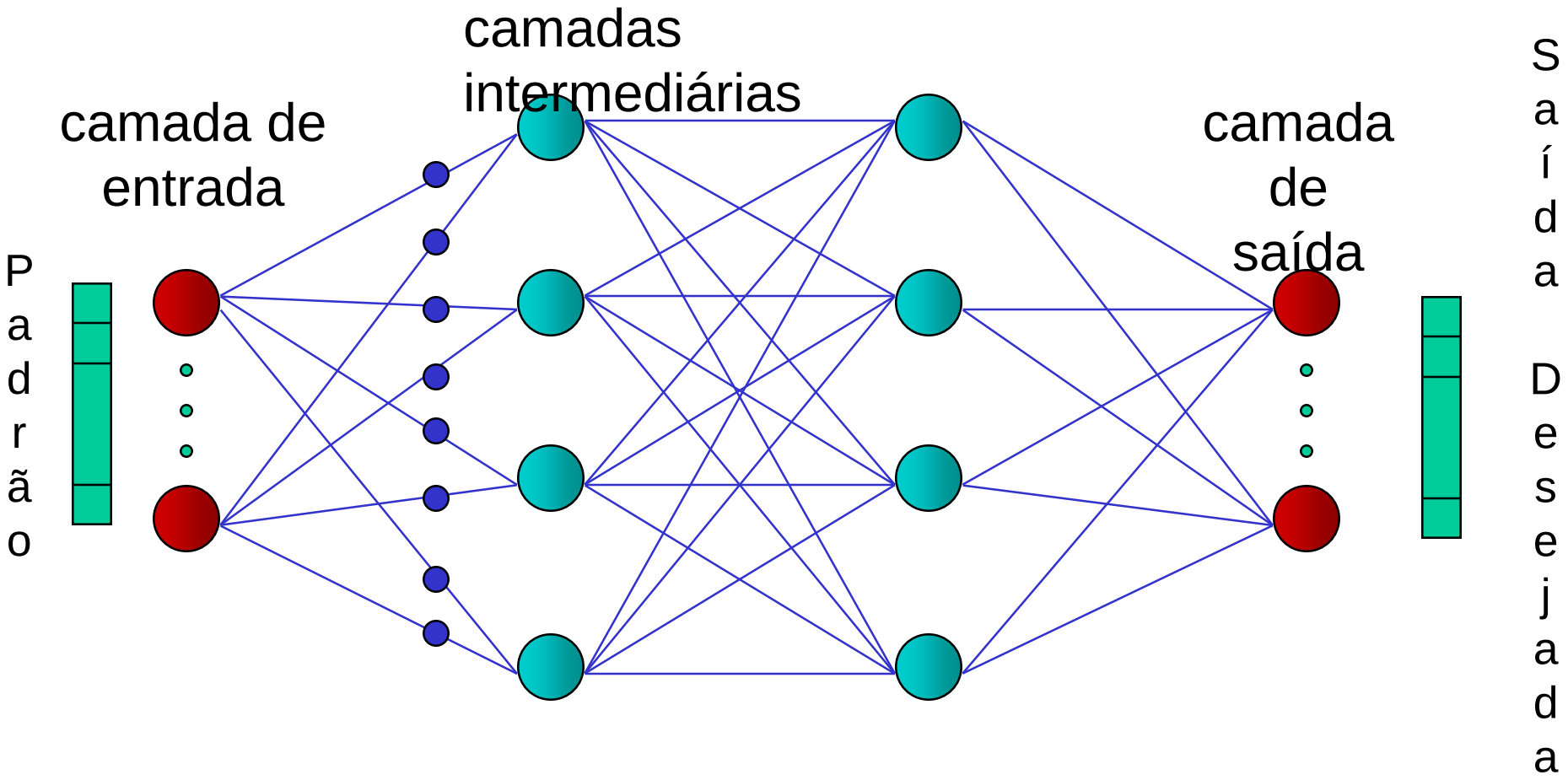
Aprendizado



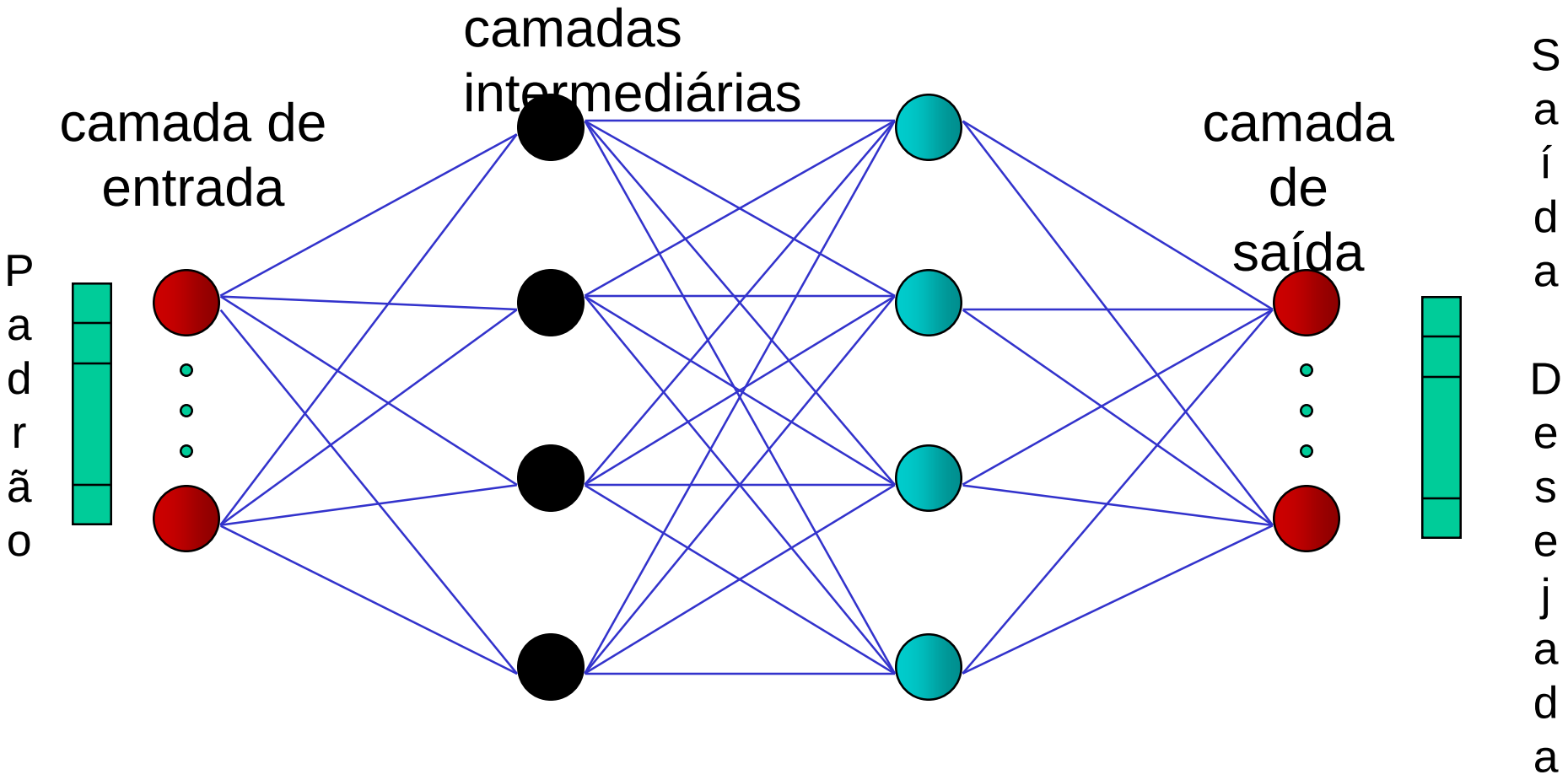
RNA - Aprendizado



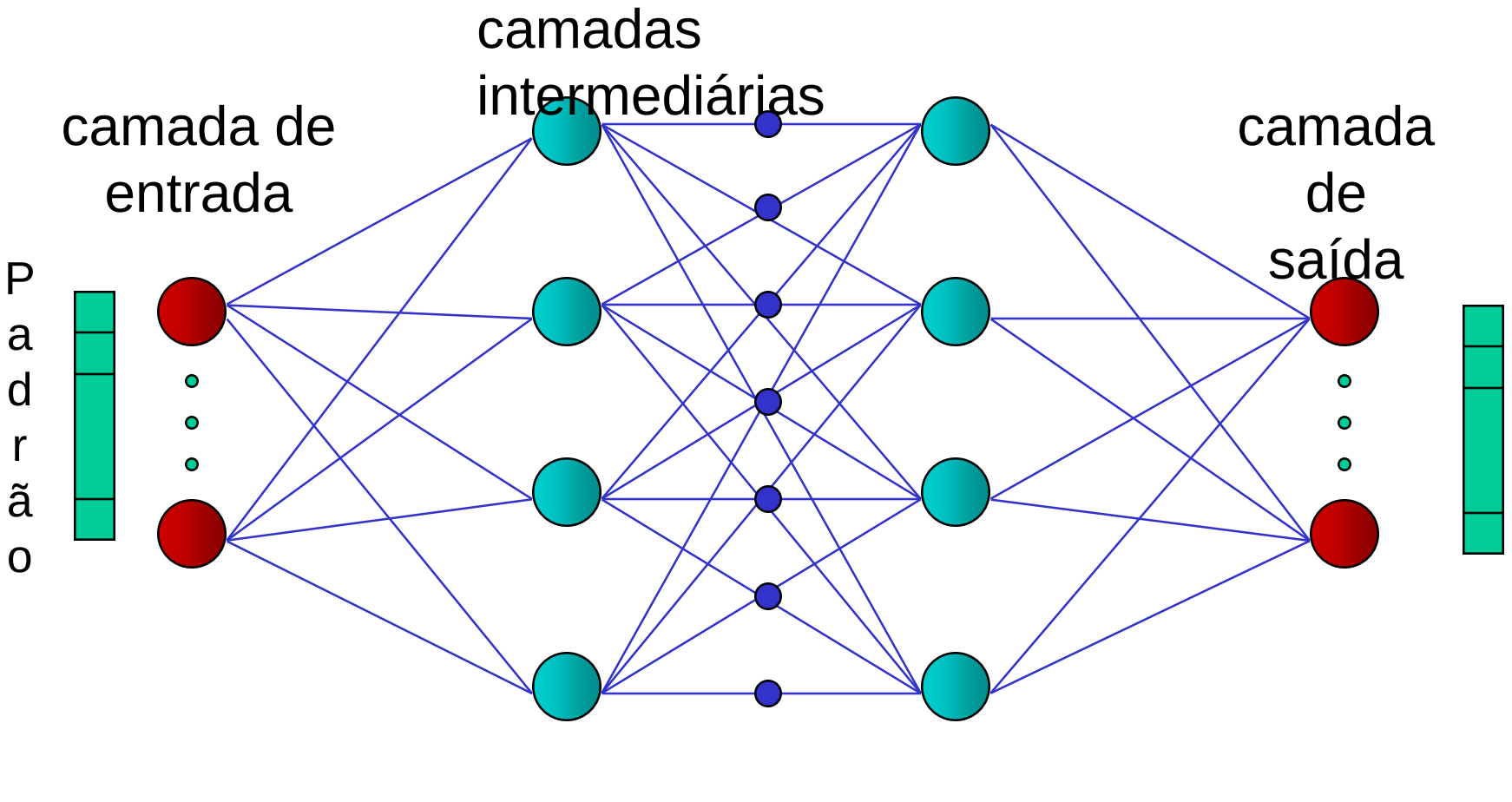
RNA - Aprendizado



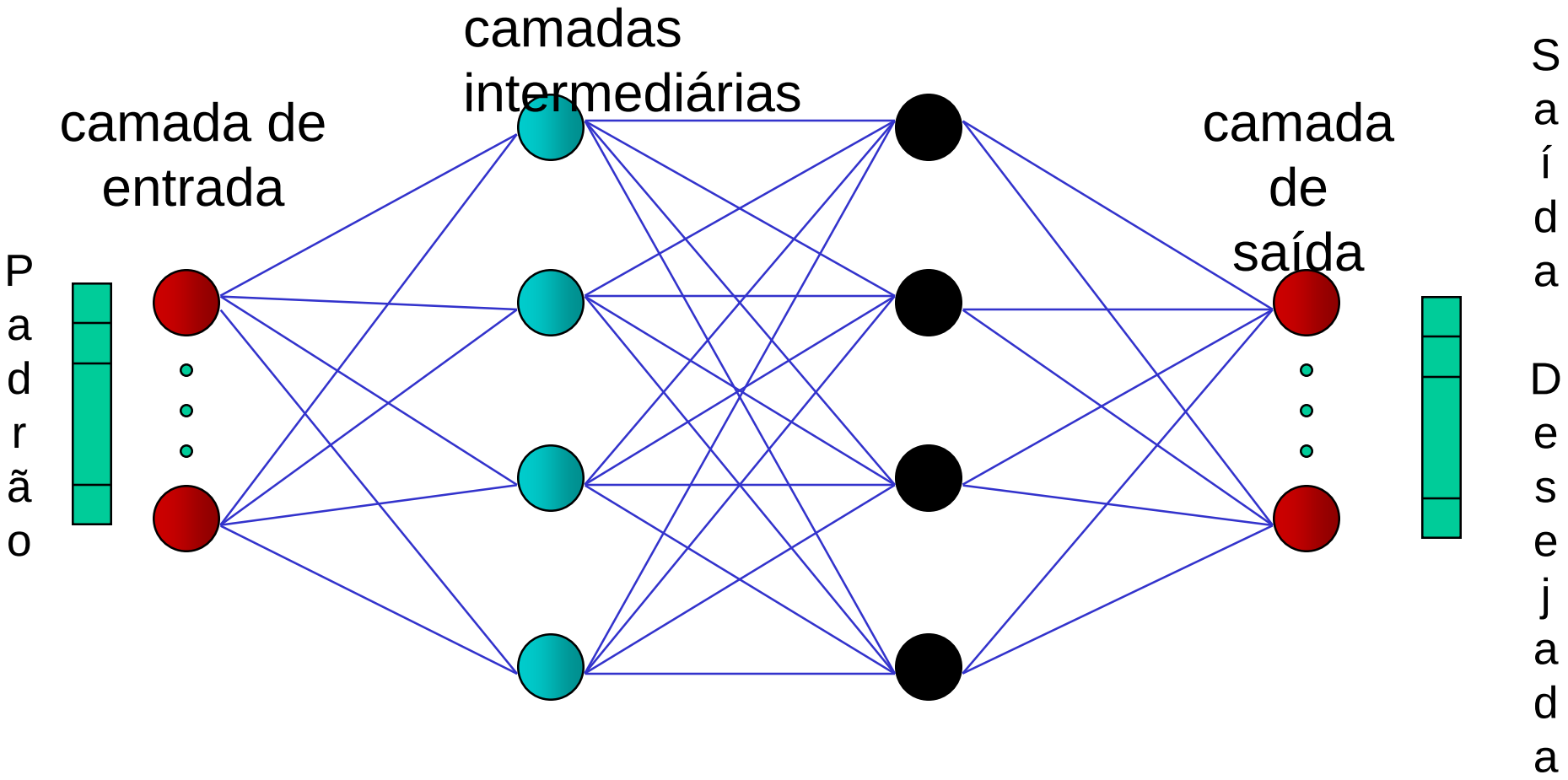
RNA - Aprendizado



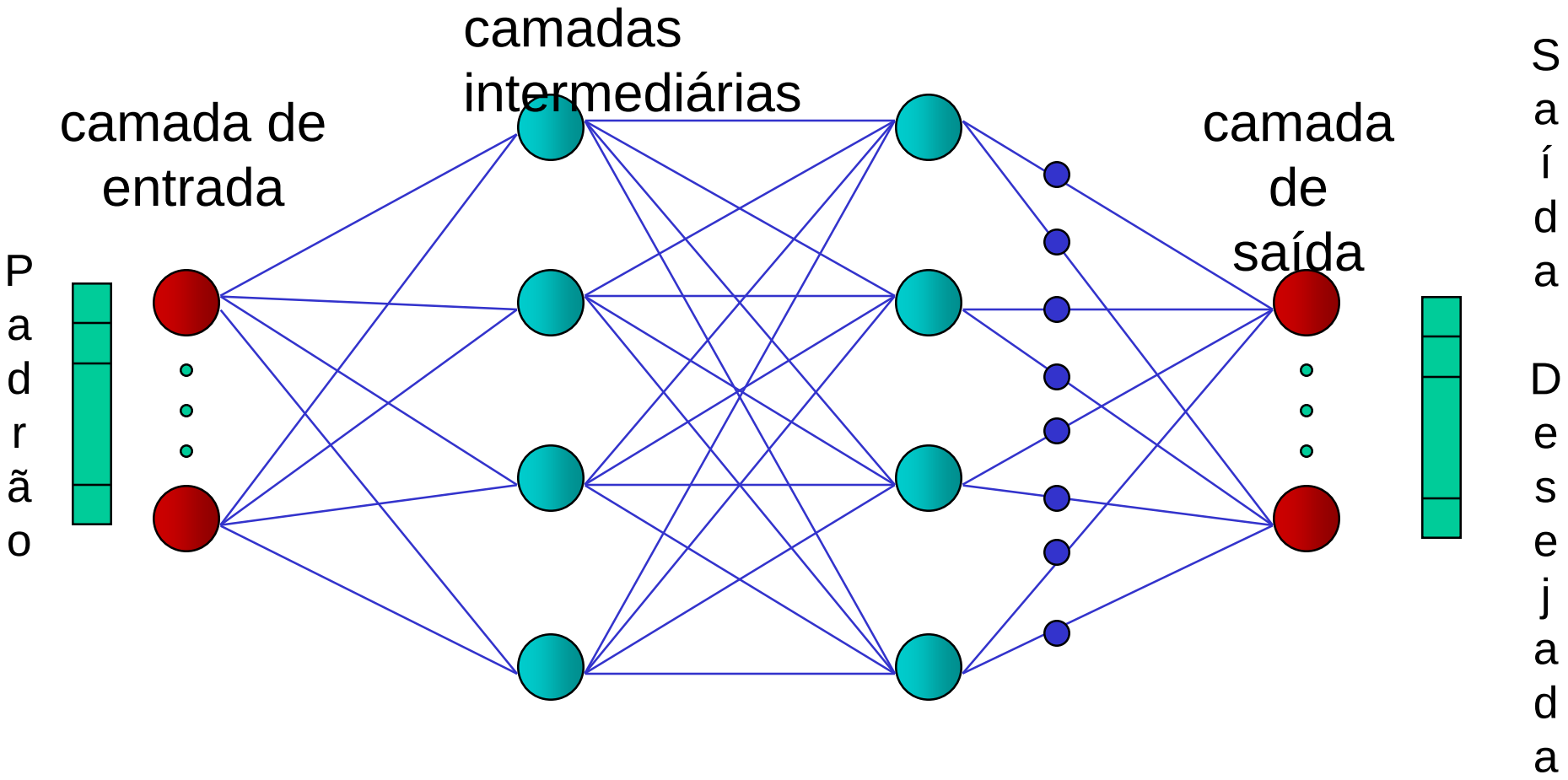
RNA - Aprendizado



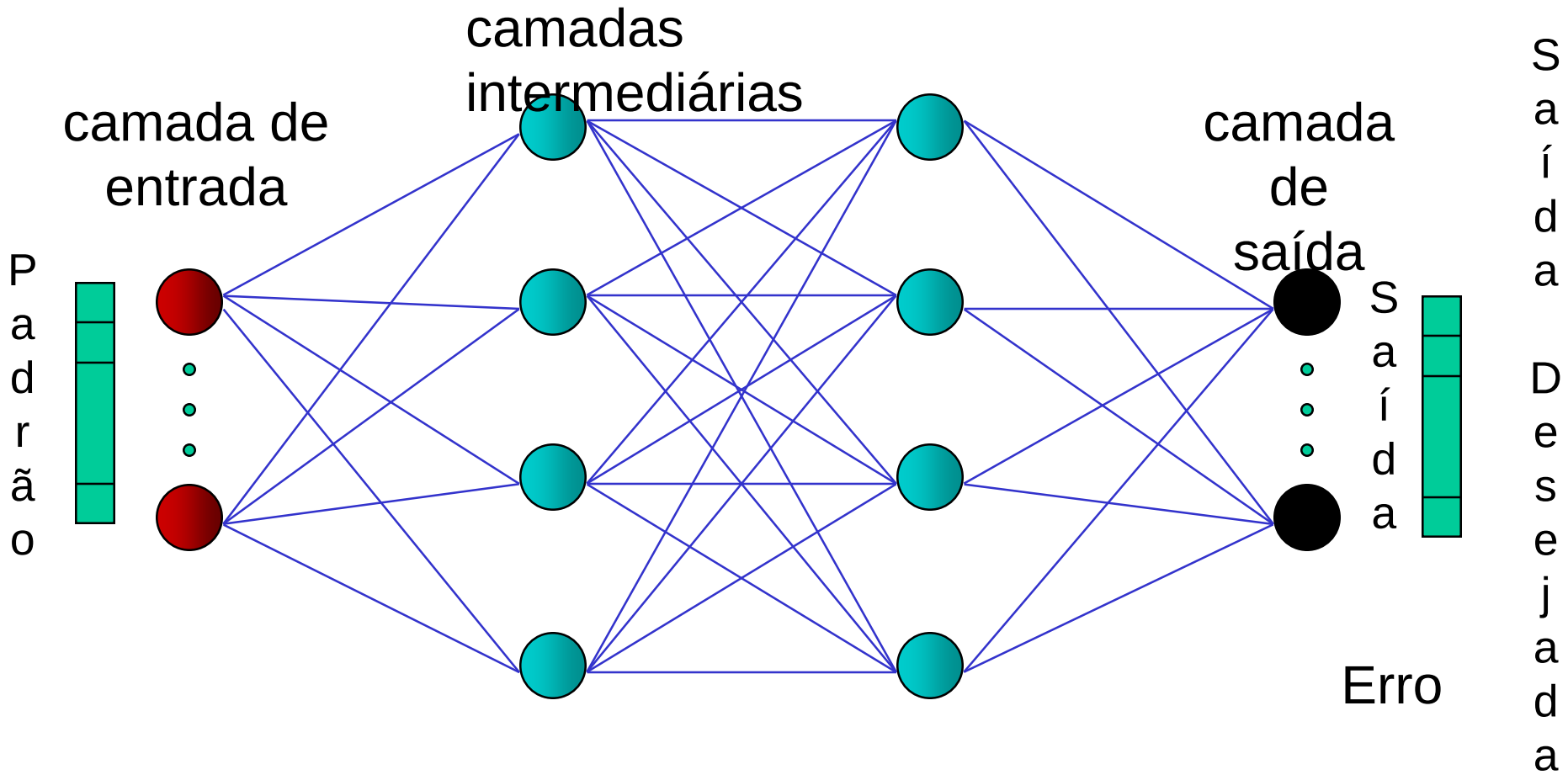
RNA - Aprendizado



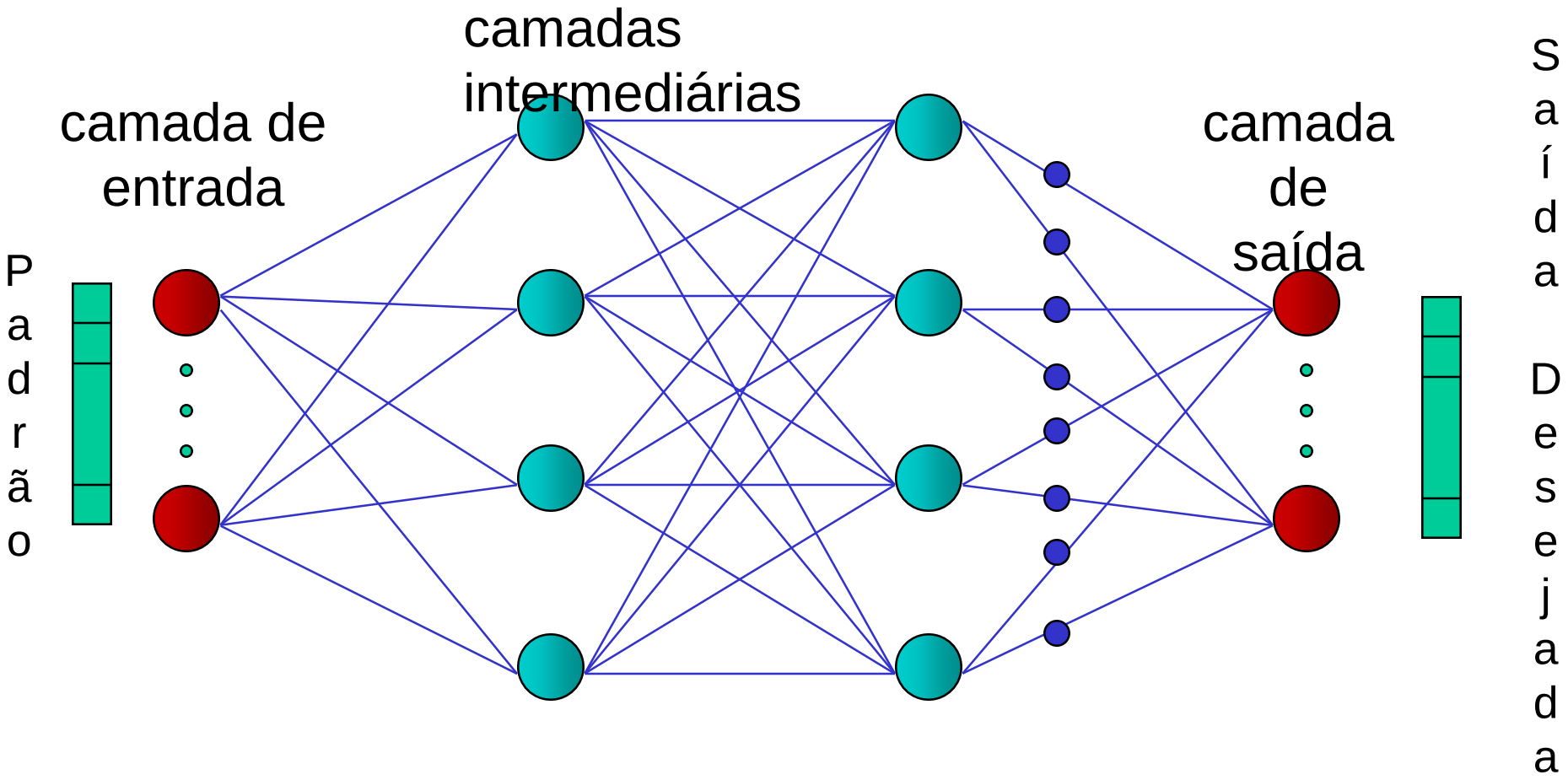
RNA - Aprendizado



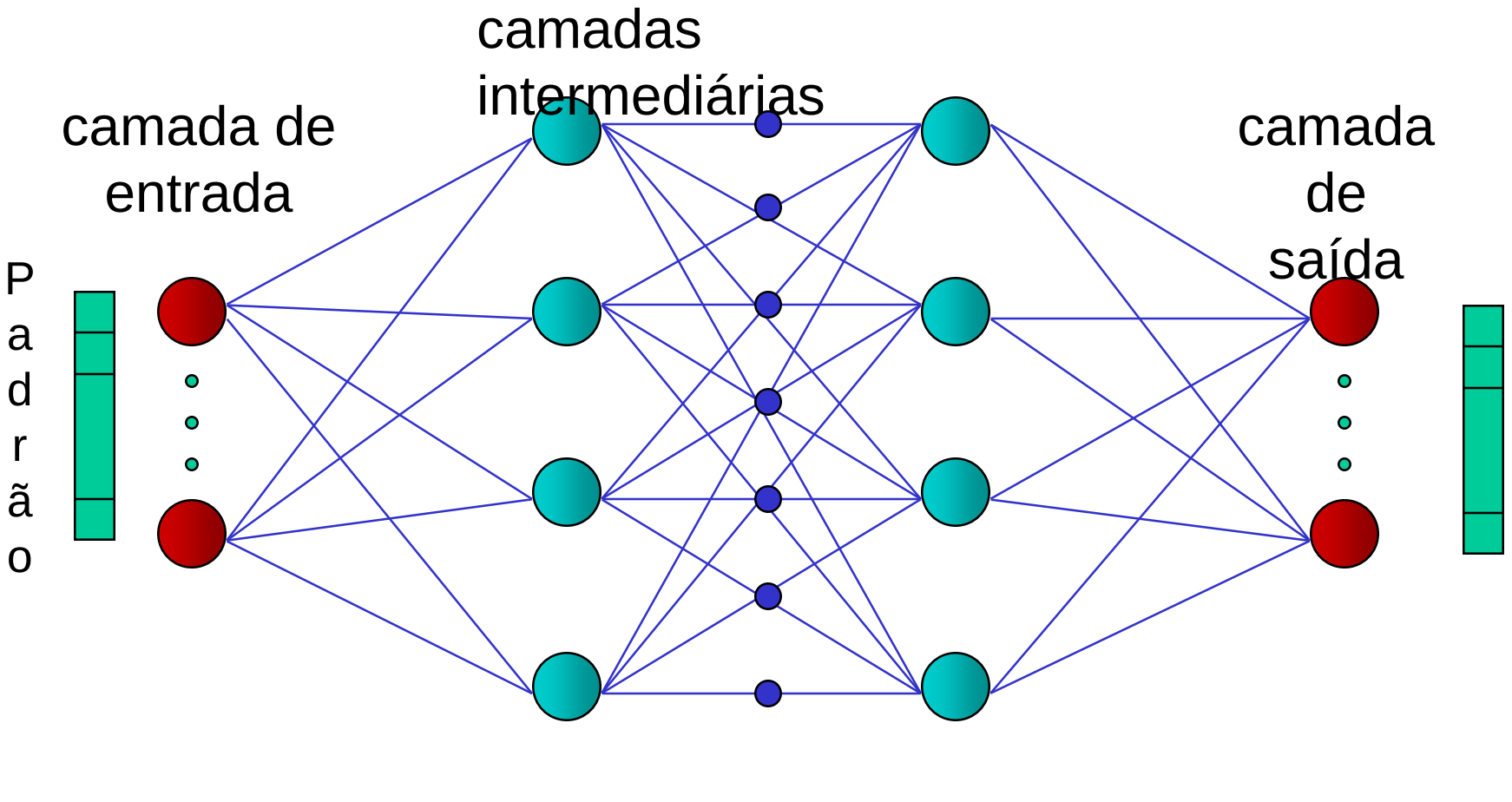
RNA - Aprendizado



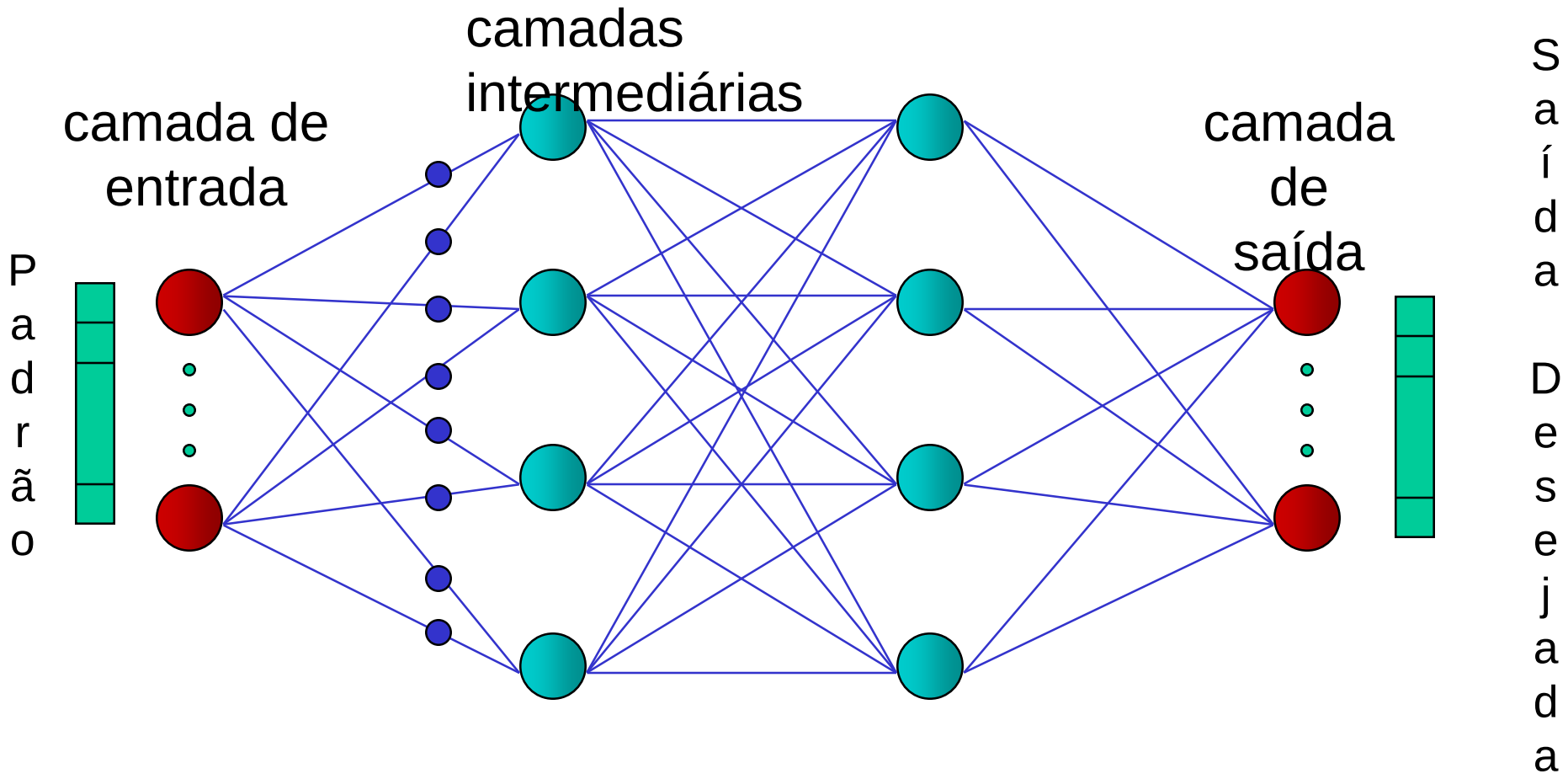
RNA - Aprendizado



RNA - Aprendizado



RNA - Aprendizado



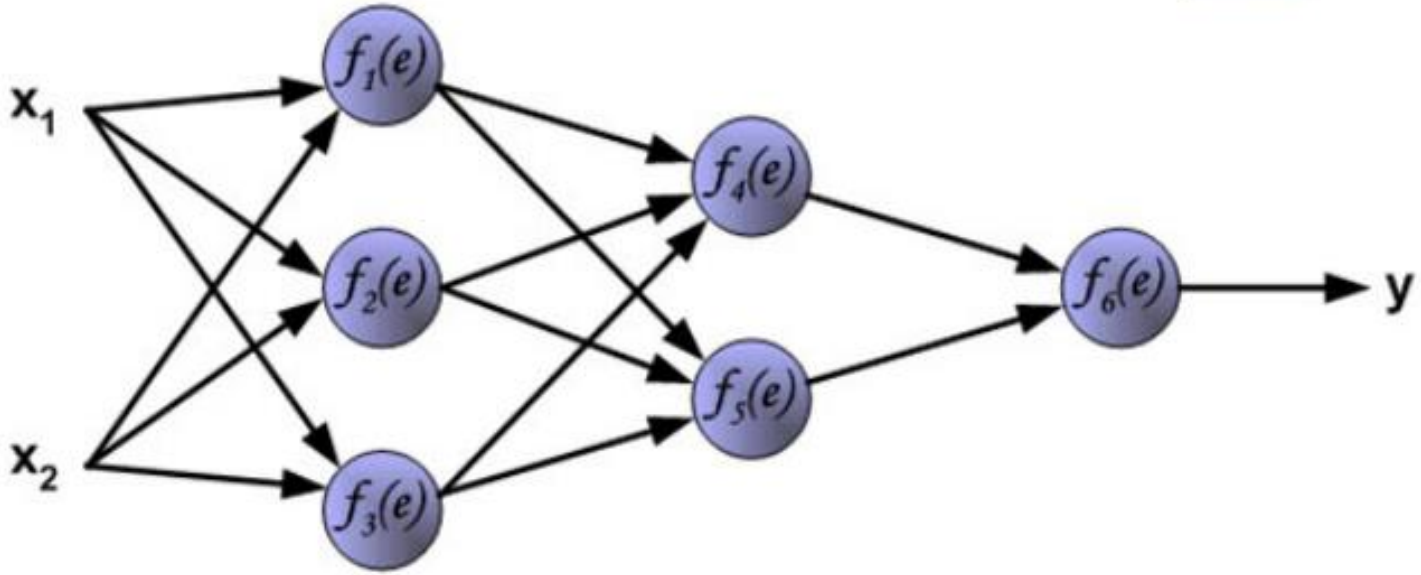
Parâmetros de treinamento

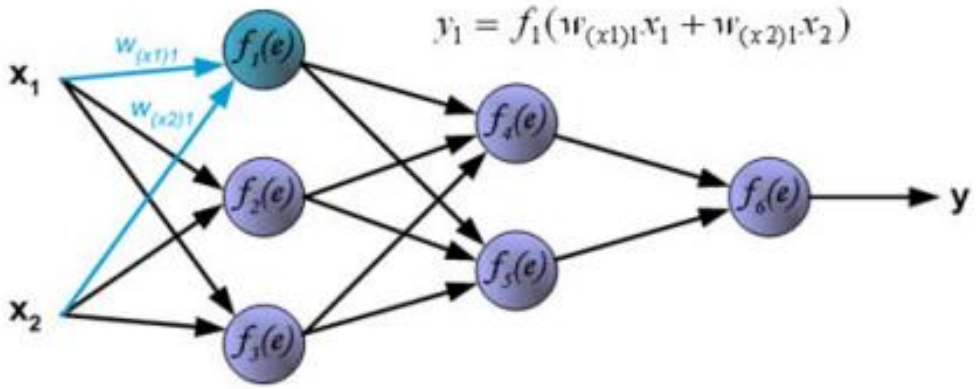
- Taxa de Aprendizado - Taxa para correção do Erro.
- Inércia - Proporcionalidade da taxa de aprendizado.
- Época - Treinamento de todas as amostras do conjunto.

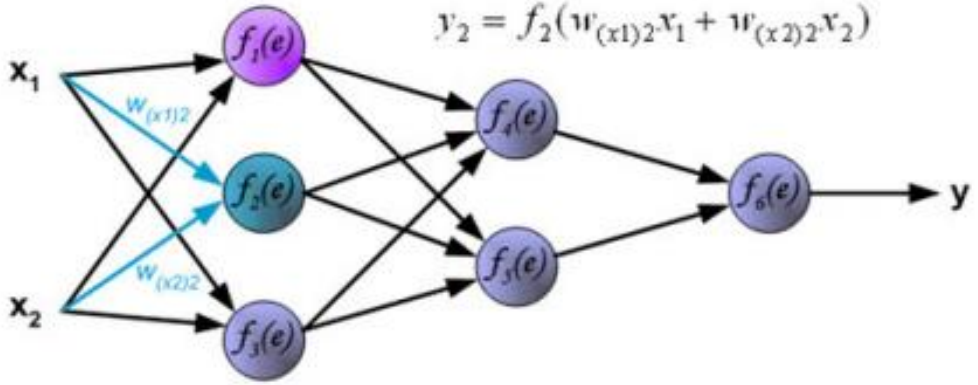
Observação sobre a regra de aprendizagem

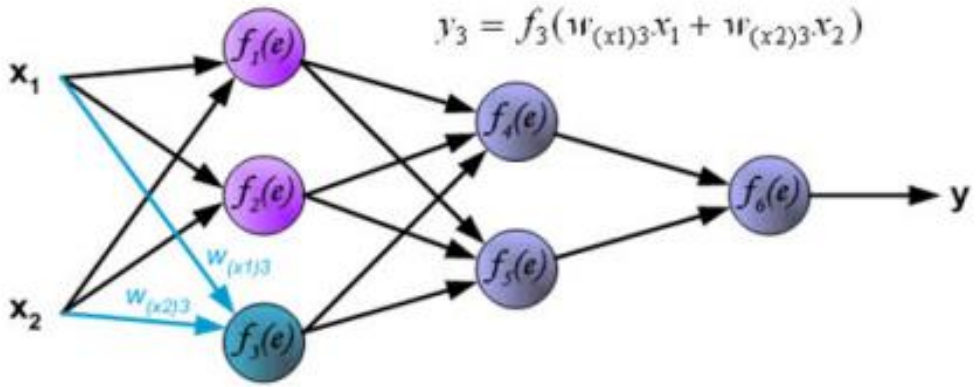
- As redes que utilizam backpropagation trabalham com uma variação da regra delta, apropriada para redes multi-camadas: a regra delta generalizada.
- A regra delta padrão essencialmente implementa um gradiente descendente no quadrado da soma do erro para funções de ativação lineares.
- Entretanto, a superfície do erro pode não ser tão simples, as redes ficam sujeitas aos problemas de de mínimos locais.

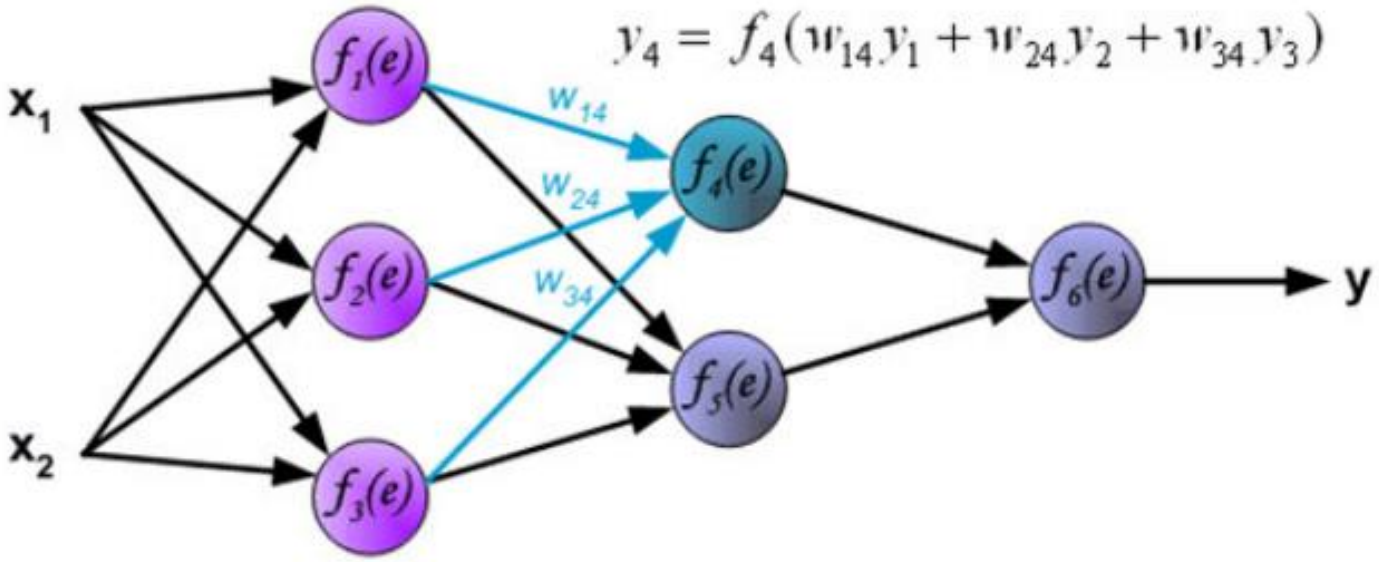
Algoritmo Backpropagation (cálculos detalhados)

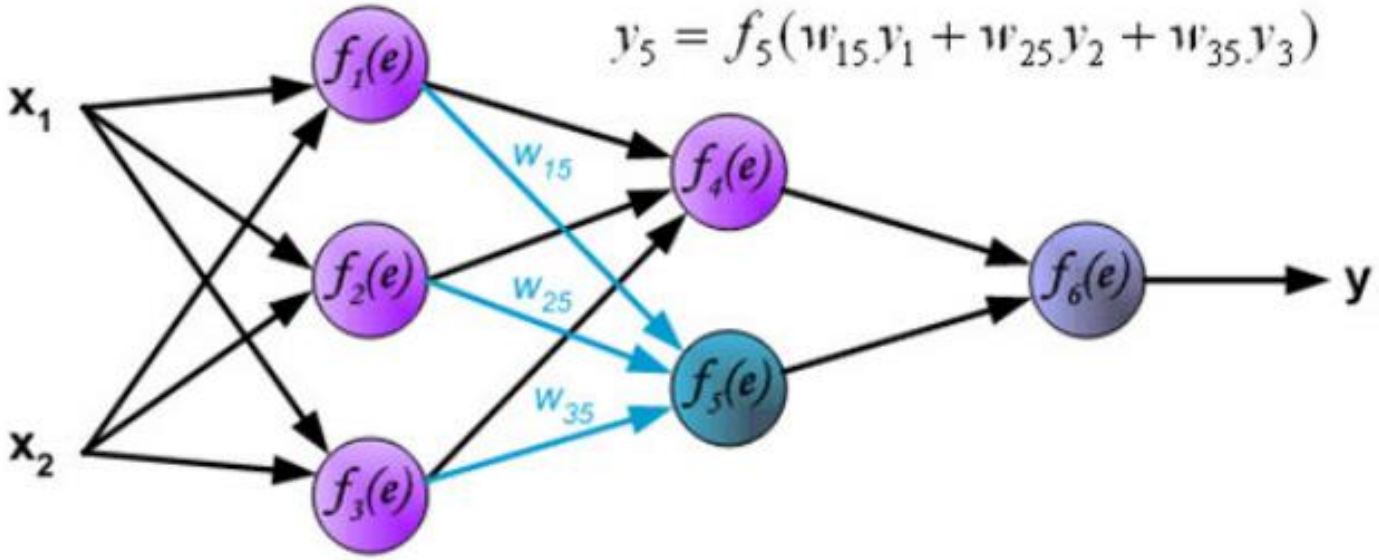


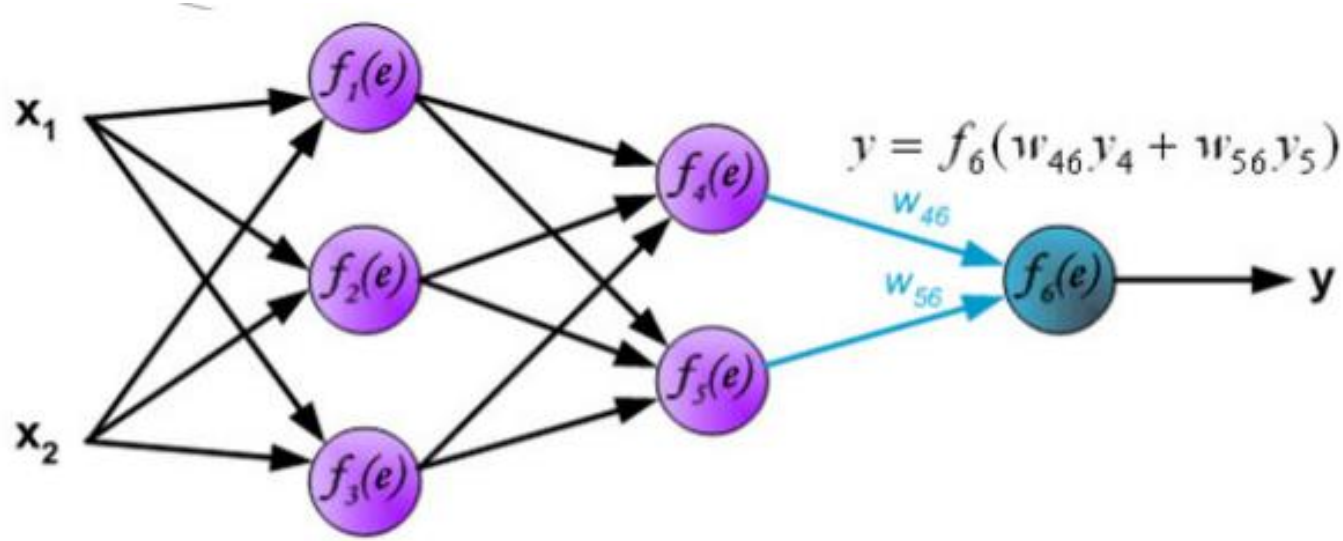

$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$
x1
x2
w(x1)1
w(x2)1
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y

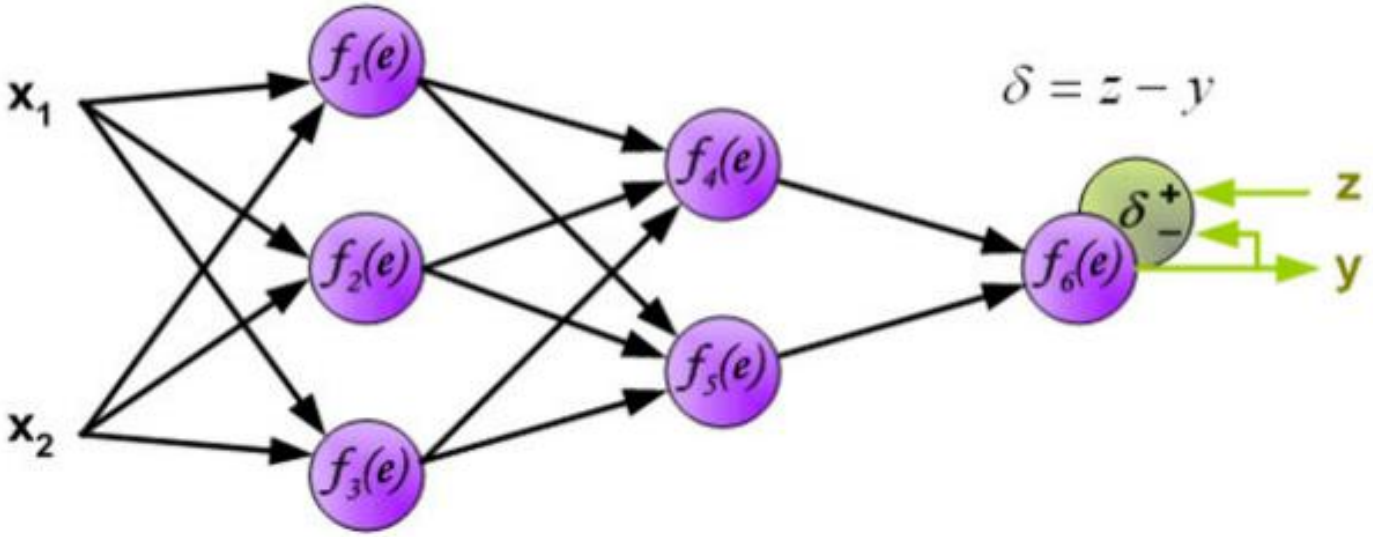

$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$
x1
x2
w(x1)2
w(x2)2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y

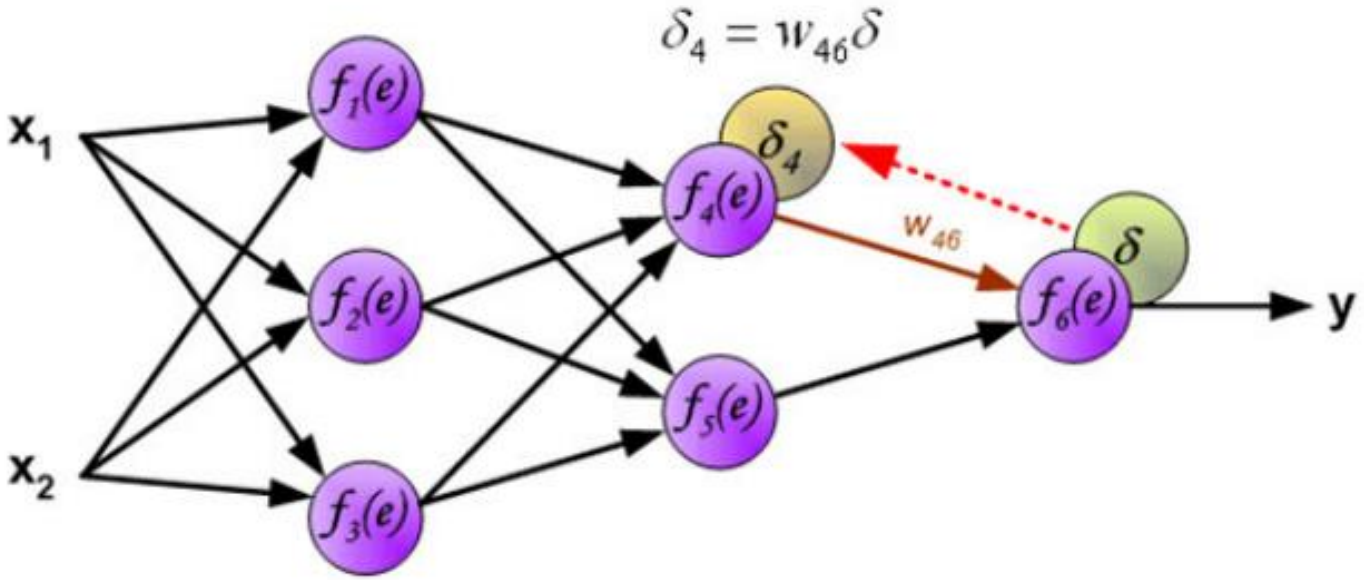

$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$
x1
x2
w(x1)3
w(x2)3
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y

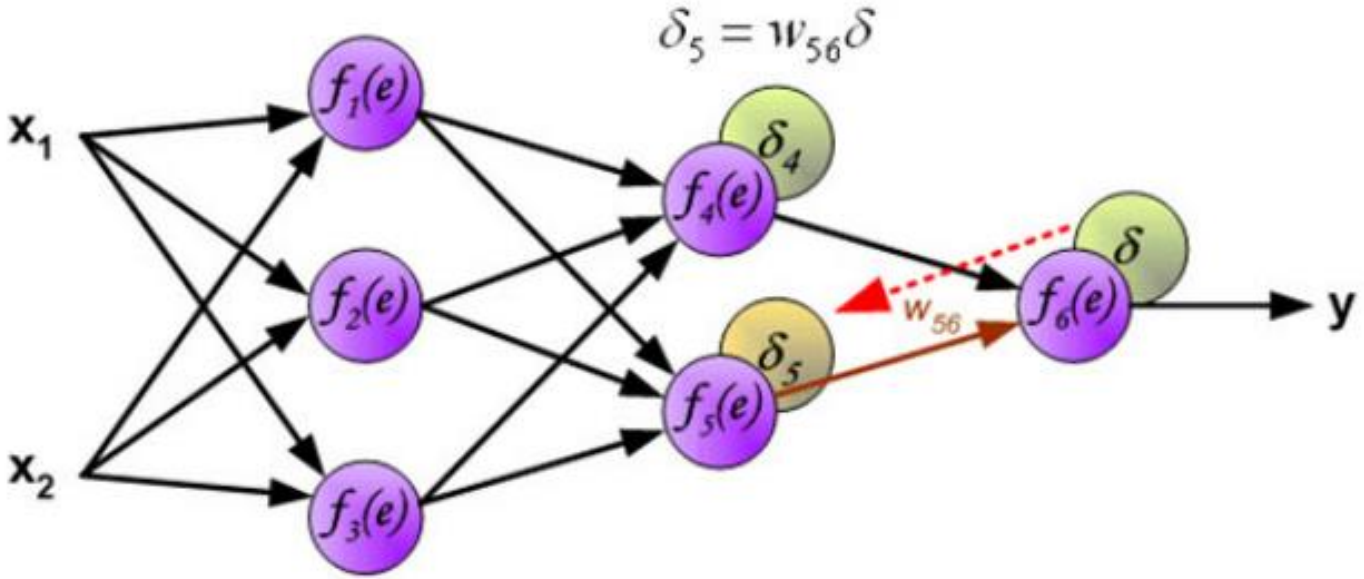

$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
w14
w24
w34
f4(e)
f5(e)
f6(e)
y

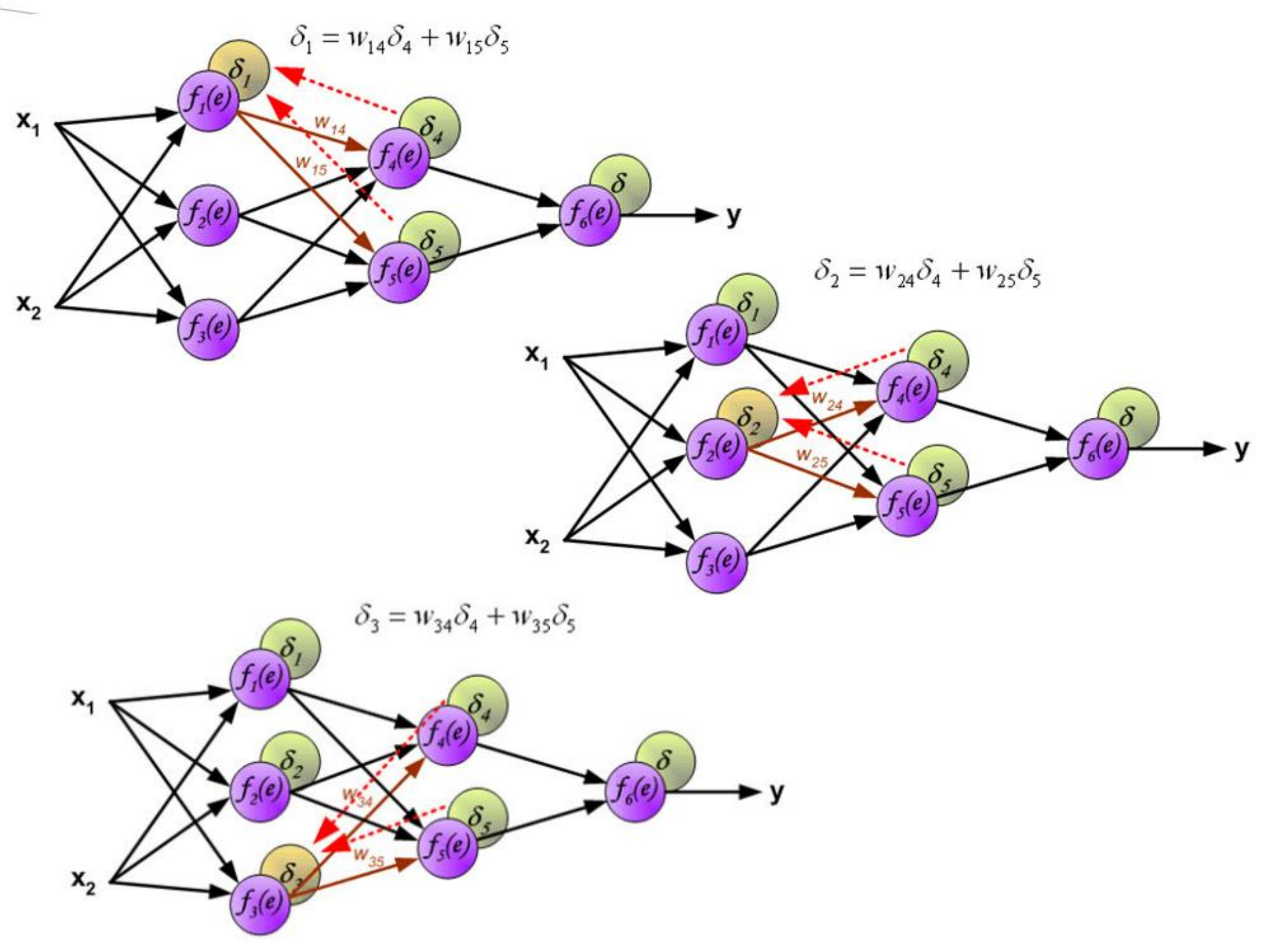

$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
w15
w25
w35
f4(e)
f5(e)
f6(e)
y


x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y = f6(w46.y4 + w56.y5)
w46
w56
y


x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
δ = z − y
δ
+
−
z
y

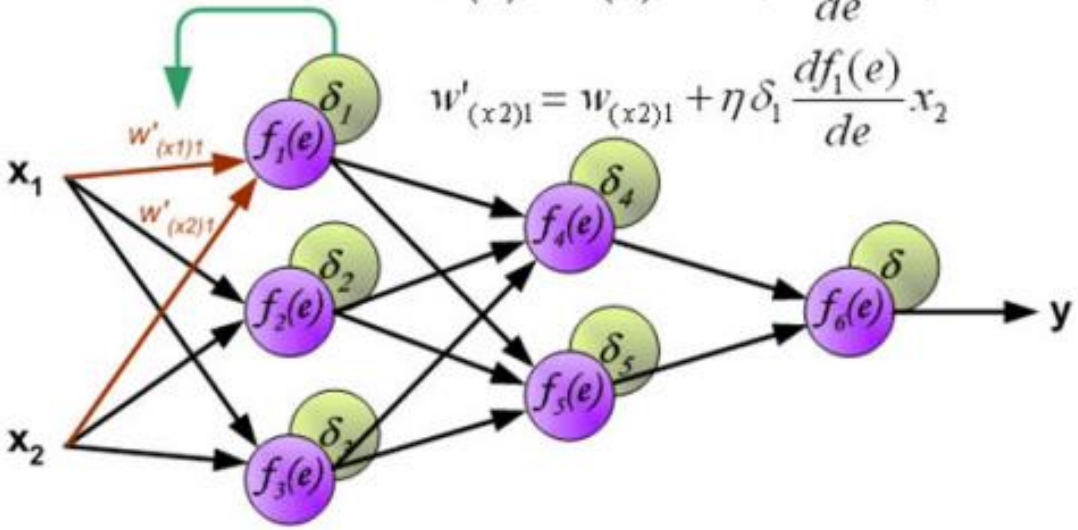

$\delta_4 = w_{46}\delta$
x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
δ4
δ
w46
y


$\delta_5 = w_{56}\delta$
x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
δ4
δ5
δ
w56
y


$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$
$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$
$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$
x1
x2
y
w14
w15
w24
w25
w34
w35
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
δ1
δ2
δ3
δ4
δ5
δ

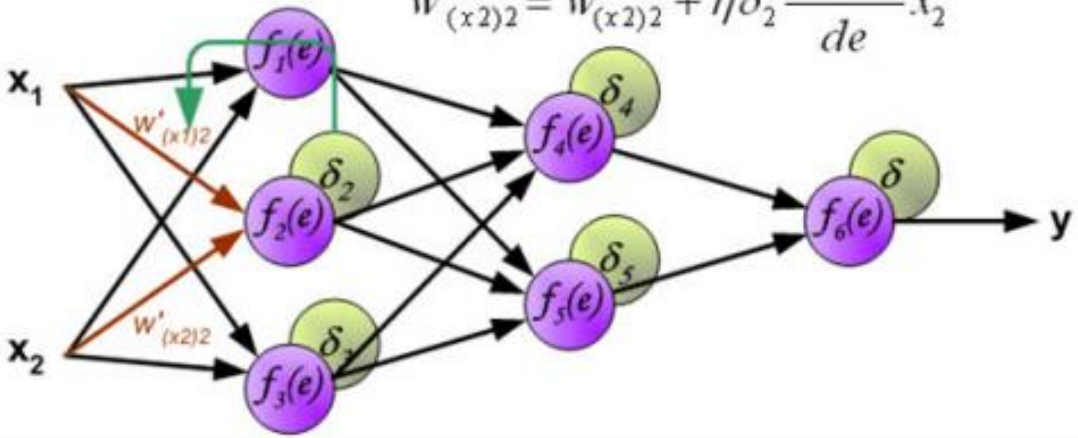
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



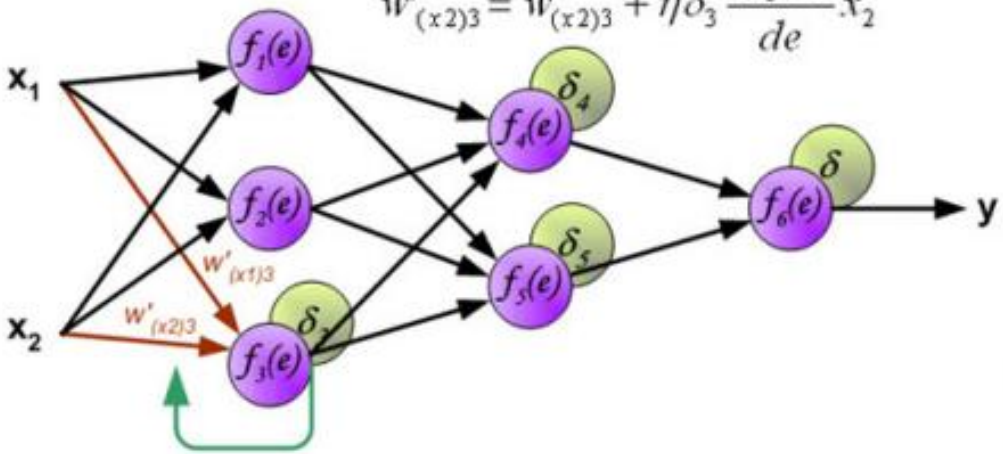
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

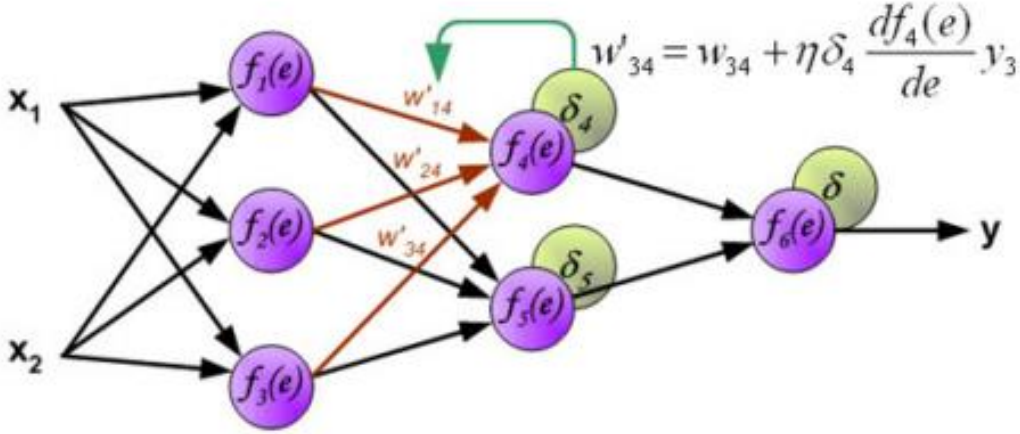
$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

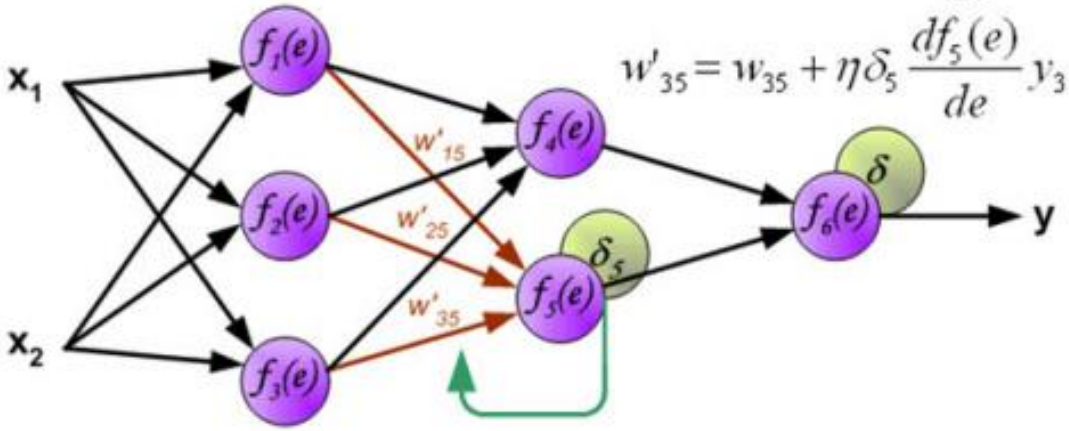
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

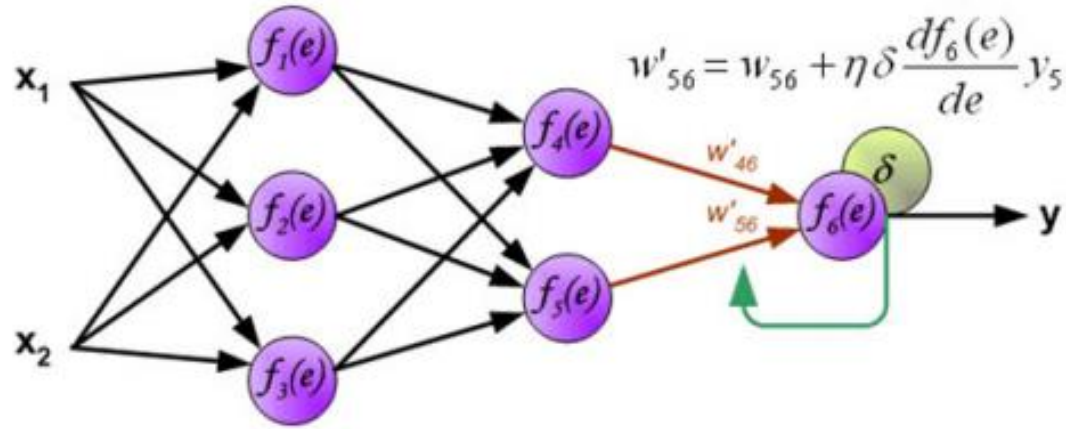
$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



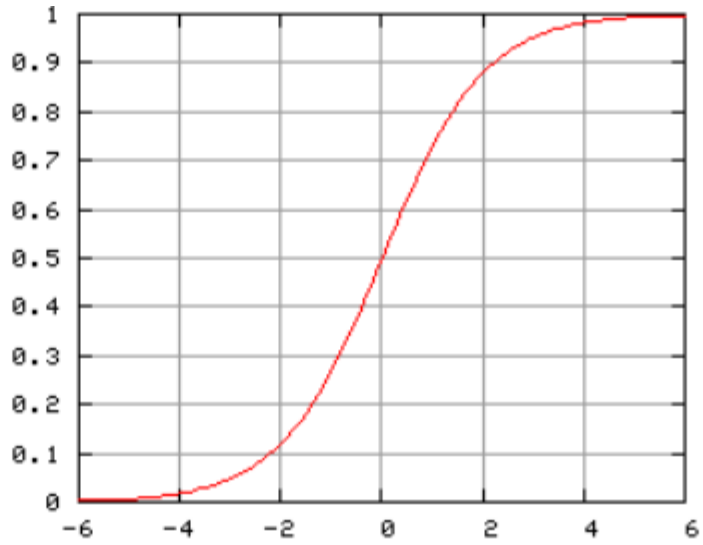
$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



Função de Ativação

Função Sigmoide

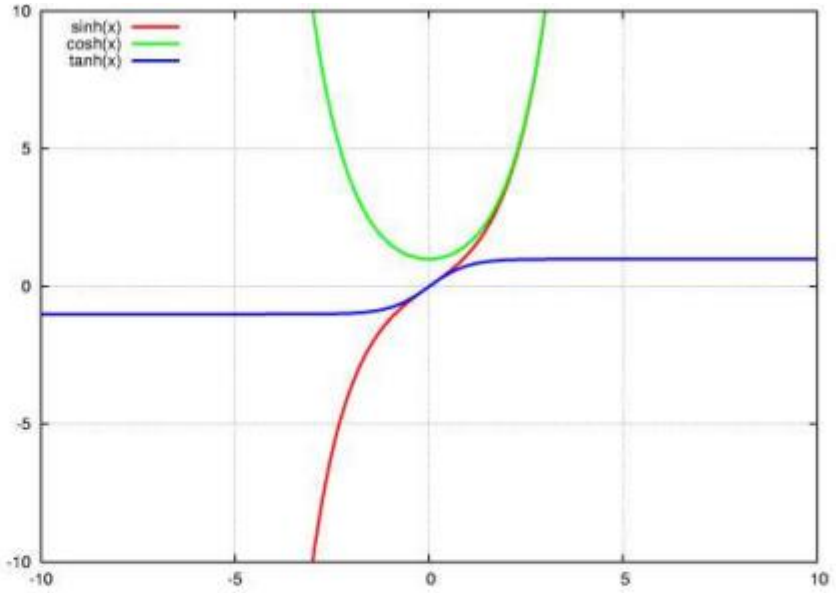


$$y_j = \varphi(net_j) = \frac{1}{1+\exp(-a \cdot net_j)} \quad a > 0 \text{ and } -\infty < net_j < \infty$$

$$\varphi'(net_j) = \frac{1}{1+\exp(-a \cdot net_j)} \frac{a \cdot [1+\exp(-a \cdot net_j) - 1]}{1+\exp(-a \cdot net_j)} = a \cdot y_j [1 - y_j]$$

Função de Ativação

- Função Tangente hiperbólica

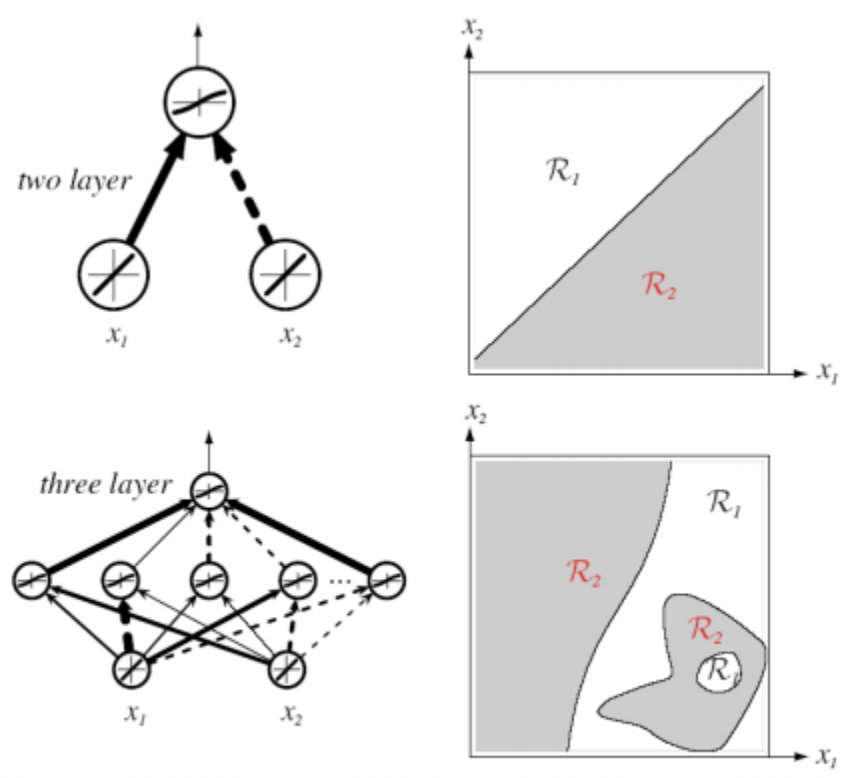


$$\varphi(net_i) = a \cdot \tanh(b \cdot net_j) \qquad (a,b) > 0$$

$$\varphi'(net_i) = a \cdot b \cdot \operatorname{sech}^2(b \cdot net_j) = a \cdot b(1 - \tanh^2(b \cdot net_j)) = \frac{b}{a}[a - y_j][a + y_j]$$

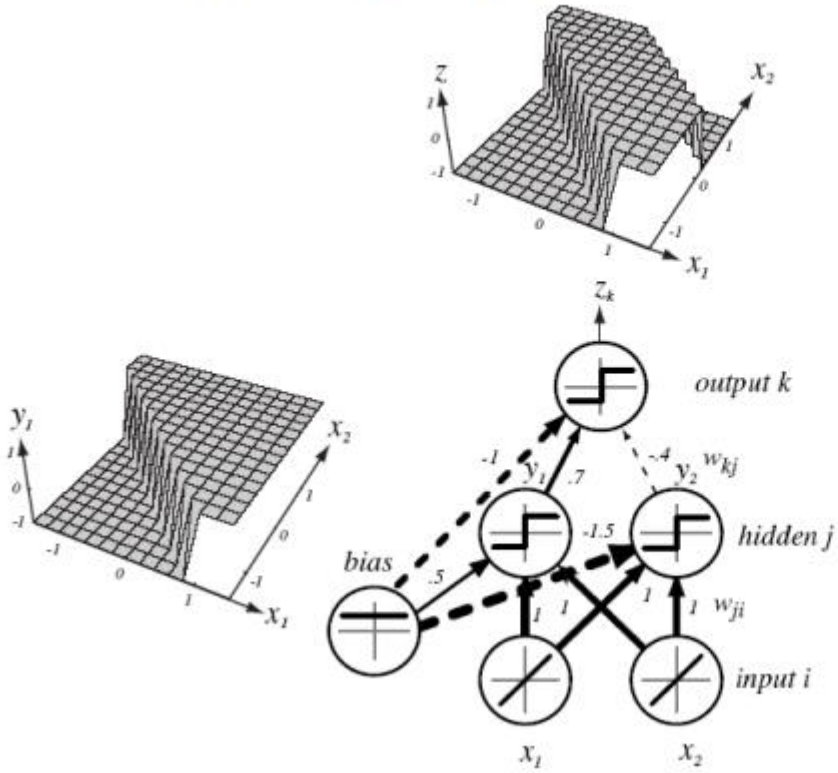
Poder Expressivo da MLP

- Qualquer transformação pode ser implementada por 3 camadas?
  - Sim, qualquer função contínua da entrada para a saída pode ser implementada com número suficiente de nodos escondidos.
  - Prova-se pelo teorema de Kolmogorov.

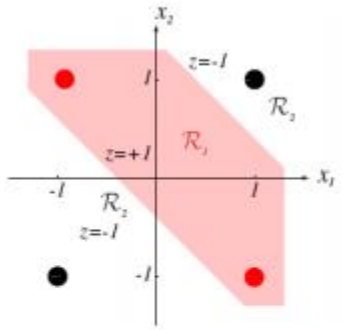


Poder Expressivo da MLP

- Porta XOR: Operação da rede

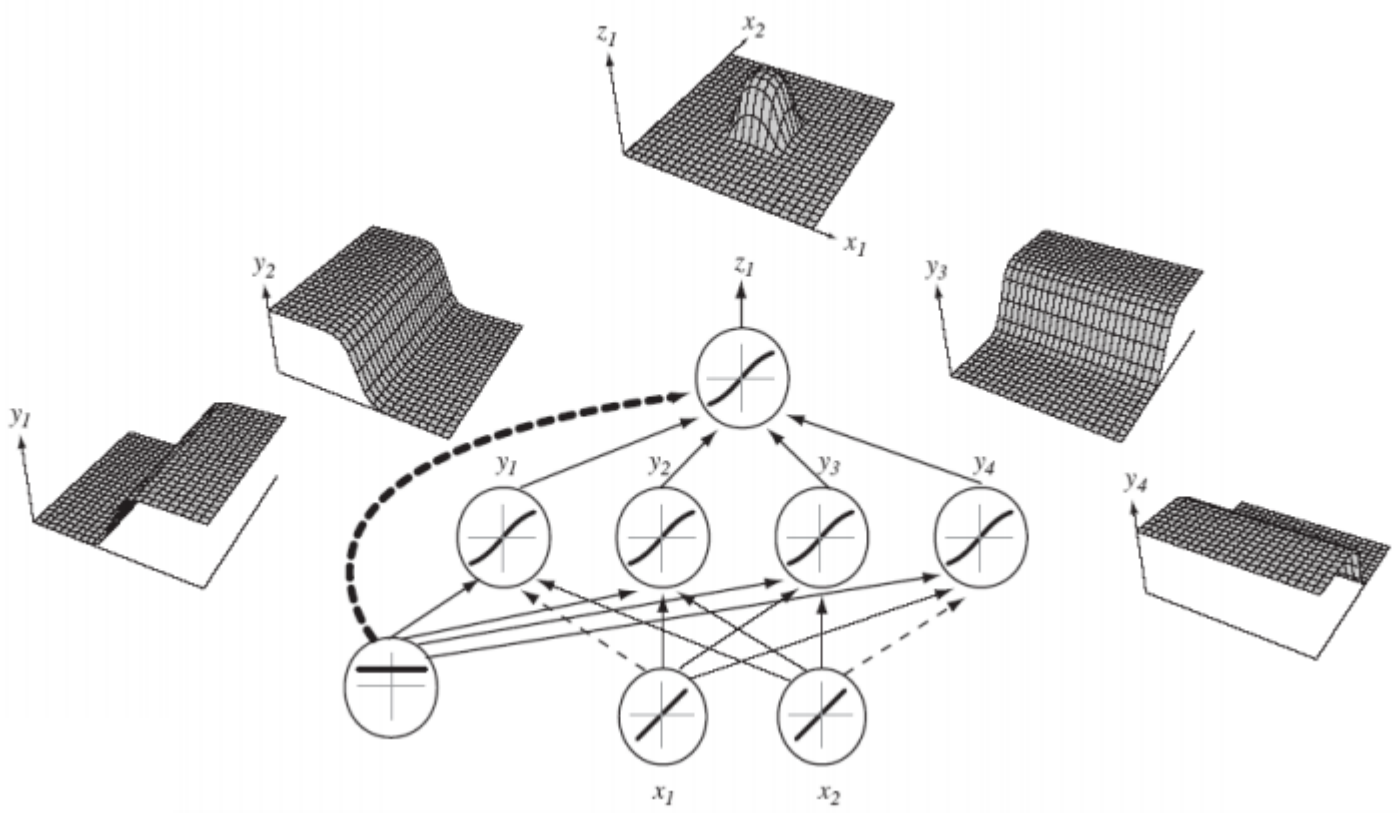


Divisão de classes

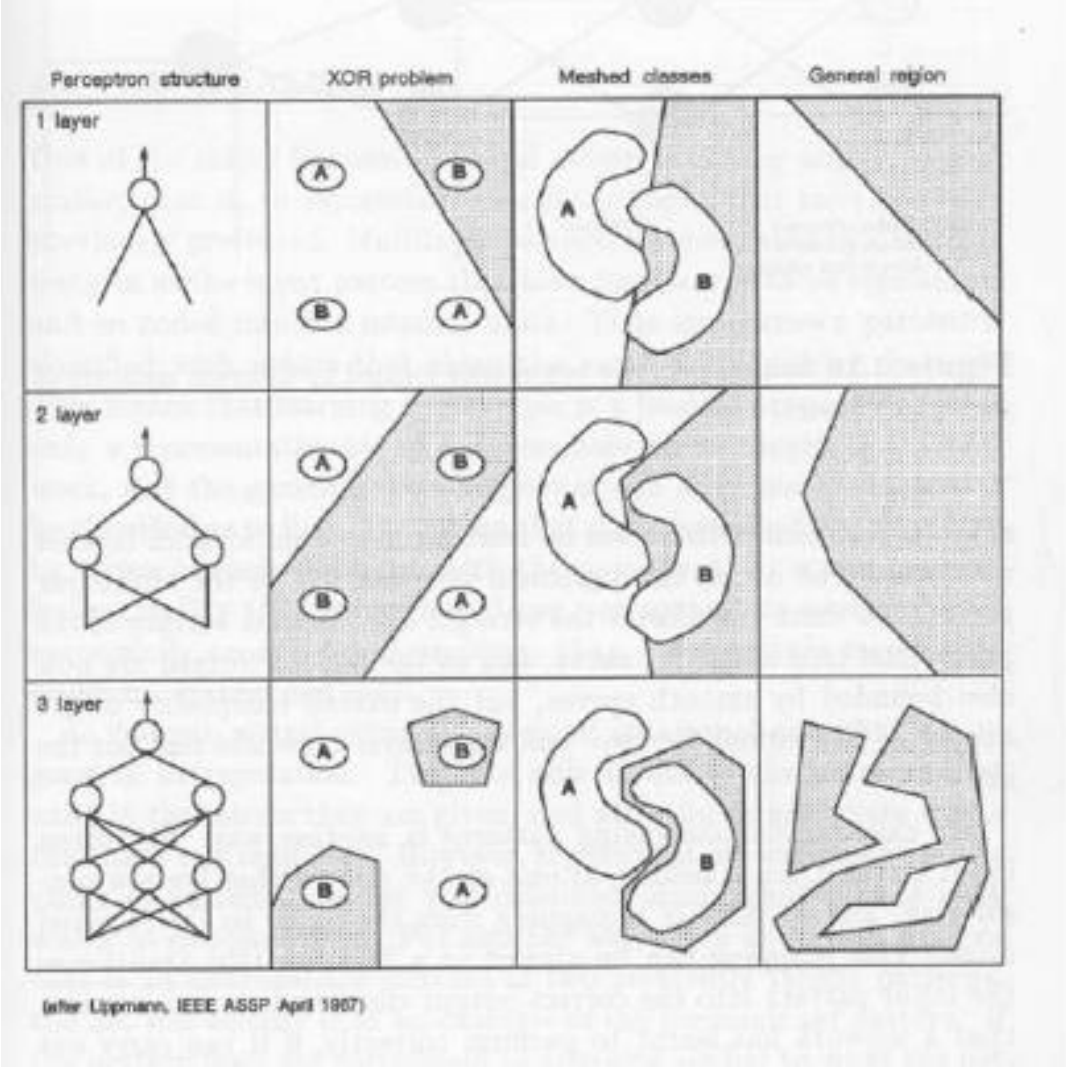


Poder Expressivo da MLP

- Exemplo de operação



Poder Expressivo da MLP



Observação sobre as câmadas escondidas

- Detector de Características:

  – Unidades escondidas atuam como detectores de características.

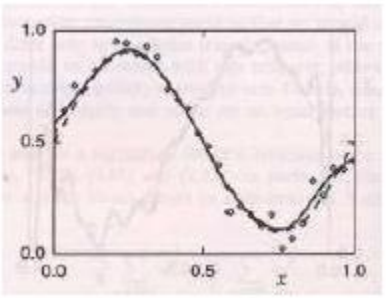
  – O progresso da aprendizagem leva as unidades escondidas a descobrirem gradualmente característica salientes dos dados de treinamento.

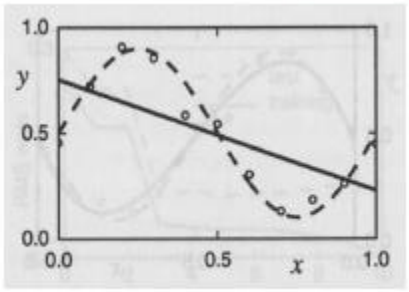
  – Transformação não-linear do espaço de entrada para o de características..

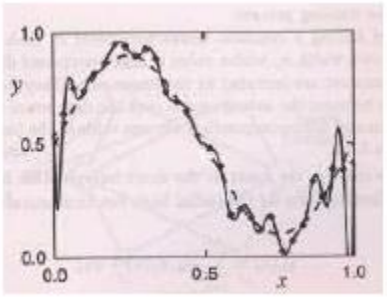
Treinamento

- Uma rede neural generaliza se seu mapeamento entrada-saída for completo ou aproximadamente correto para os dados de teste.
  - – O processo de treinamento pode ser visto como a solução de um problema de ajuste de curva para interpolação não-linear dos dados de entrada.
  - – Uma RN deixa de generalizar quando está sobretreinada, isto é, quando aprende muitos pares entrada-saída e passa a considerar características não gerais para o conjunto de padrões.

Complexidade funcional versus overfitting







Conclusão

Necessidade de redes multicamadas

- Redes de uma camada não resolvem problemas não linearmente separáveis
- Problema: treinamento de redes multicamadas
- Backpropagation
  - Overfitting
  - Momento