Mini-projeto: Reconhecimento de Digitos com MLPs

Germano C. Vasconcelos
Centro de Informática – UFPE

Link do Material

http://neuralnetworksanddeeplearning.com/chap1.html

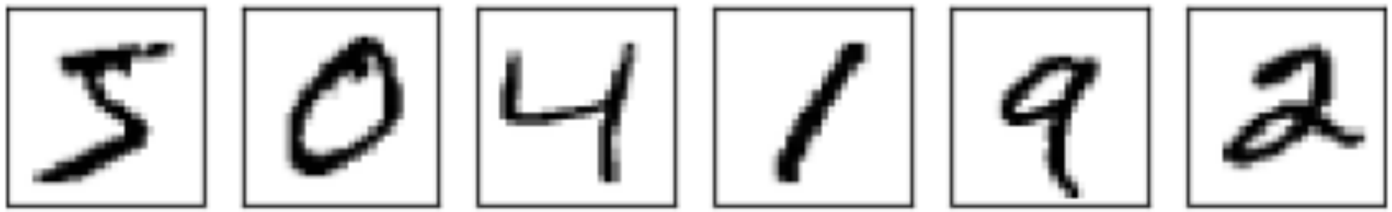
Implementing our network to classify digits

Alright, let's write a program that learns how to recognize handwritten digits, using stochastic gradient descent and the MNIST training data. We'll do this with a short Python (2.7) program, just 74 lines of code! The first thing we need is to get the MNIST data. If you're a `git` user then you can obtain the data by cloning the code repository for this book,

```
git clone https://github.com/mnielsen/neural-networks-and-deep-learning.git
```

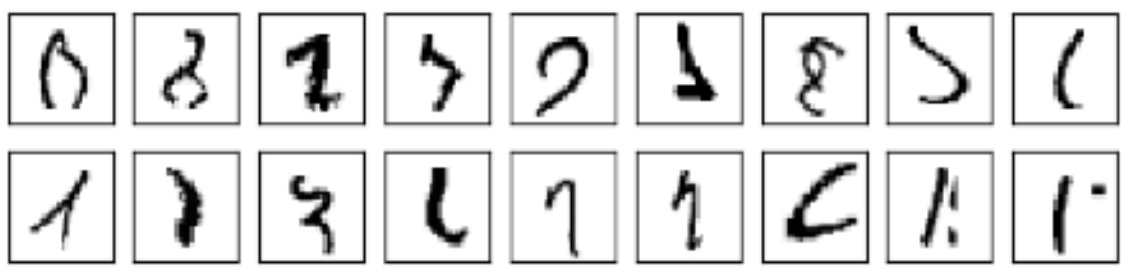
If you don't use `git` then you can download the data and code here.

Objetivo: Treinar uma Rede MLP para reconhecer digitos: MNIST



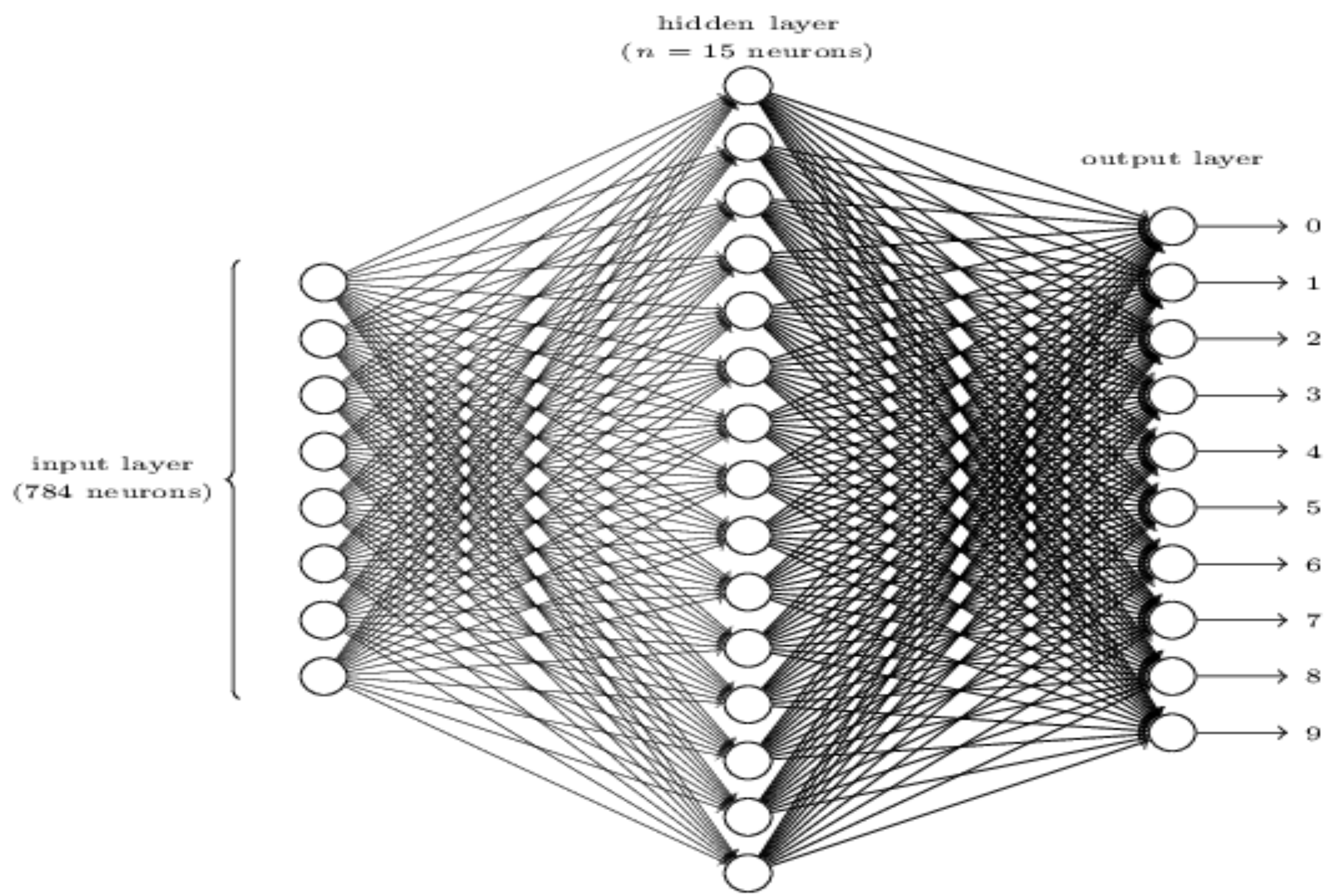
Easy?!

Not that fast! →



Objetivo: Treinar uma Rede MLP para reconhecer digitos: MNIST

To recognize individual digits we will use a three-layer neural network:



Com o Quê?

Código python, passo a passo, e base de dados dividida em treinamento e teste

How well does the program recognize handwritten digits? Well, let's start by loading in the MNIST data. I'll do this using a little helper program, `mnist_loader.py`, to be described below. We execute the following commands in a Python shell,

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
```

Of course, this could also be done in a separate Python program, but if you're following along it's probably easiest to do in a Python shell.

After loading the MNIST data, we'll set up a `Network` with 30 hidden neurons. We do this after importing the Python program listed above, which is named `network`,

Base de Dados

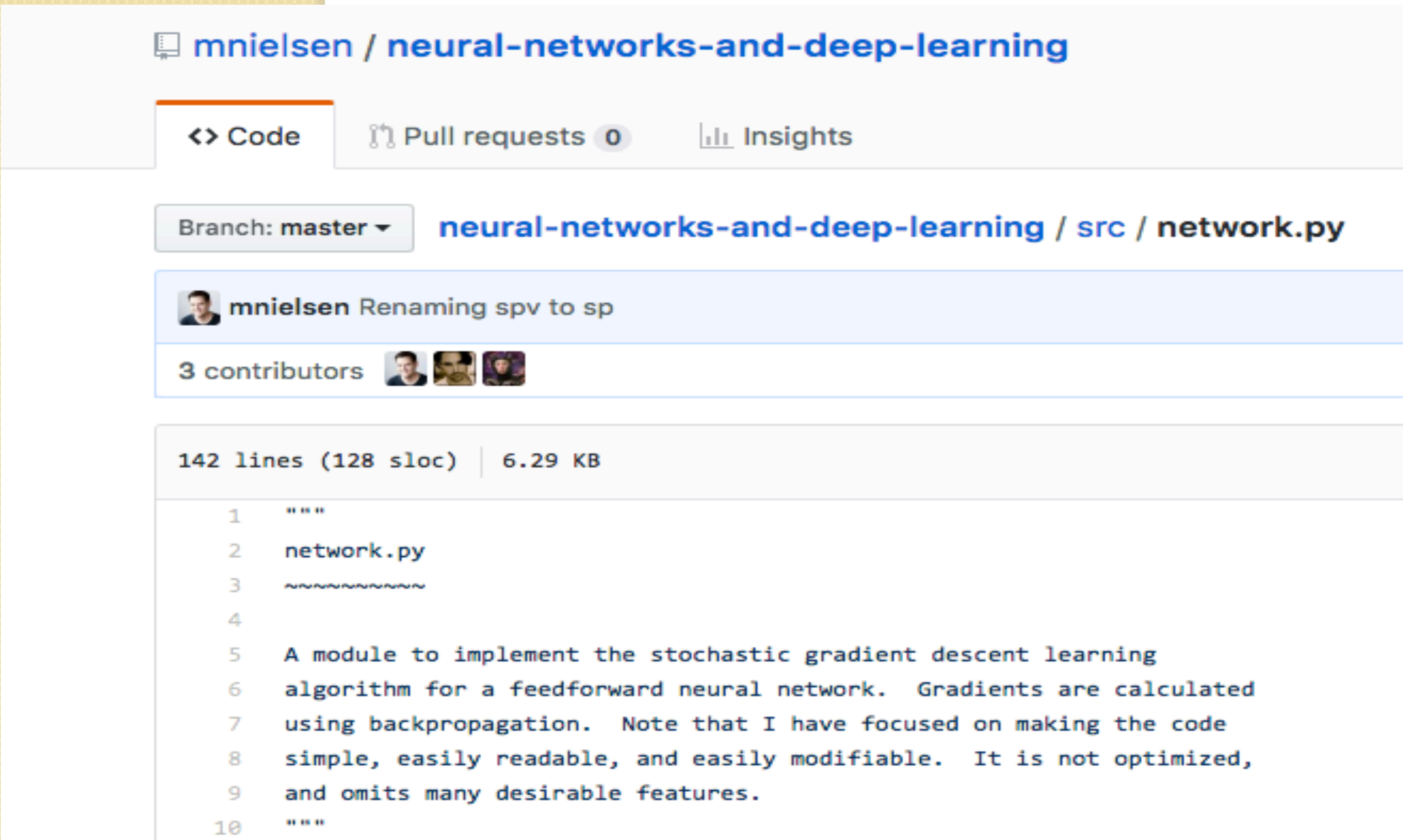
THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

Código no GitHub



Algumas Informações

Apart from the MNIST data we also need a Python library called Numpy, for doing fast linear algebra. If you don't already have Numpy installed, you can get it here.

Algumas Informações

How well does the program recognize handwritten digits? Well, let's start by loading in the MNIST data. I'll do this using a little helper program, `mnist_loader.py`, to be described below. We execute the following commands in a Python shell,

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
```

Of course, this could also be done in a separate Python program, but if you're following along it's probably easiest to do in a Python shell.

After loading the MNIST data, we'll set up a `Network` with 30 hidden neurons. We do this after importing the Python program listed above, which is named `network`,

```
>>> import network
>>> net = network.Network([784, 30, 10])
```

Finally, we'll use stochastic gradient descent to learn from the MNIST `training_data` over 30 epochs, with a mini-batch size of 10, and a learning rate of $\eta = 3.0$,

Algumas Informações

How well does the program recognize handwritten digits? Well, let's start by loading in the MNIST data. I'll do this using a little helper program, `mnist_loader.py`, to be described below. We execute the following commands in a Python shell,

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
```

Of course, this could also be done in a separate Python program, but if you're following along it's probably easiest to do in a Python shell.

After loading the MNIST data, we'll set up a `Network` with 30 hidden neurons. We do this after importing the Python program listed above, which is named `network`,

```
>>> import network
>>> net = network.Network([784, 30, 10])
```

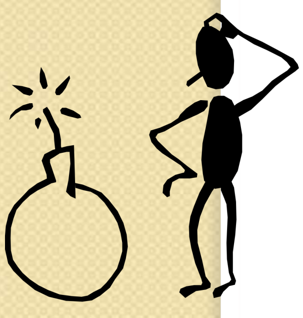
Finally, we'll use stochastic gradient descent to learn from the MNIST `training_data` over 30 epochs, with a mini-batch size of 10, and a learning rate of $\eta = 3.0$,

O Que Entregar?

- Taxas de acerto (acurácia) por classe e total
- Pode ser o código mostrando execução, passo a passo, com as taxas de acerto ao final
- Precisa evidenciar a execução!
- **Prazo: 2 semanas (sharp!)**

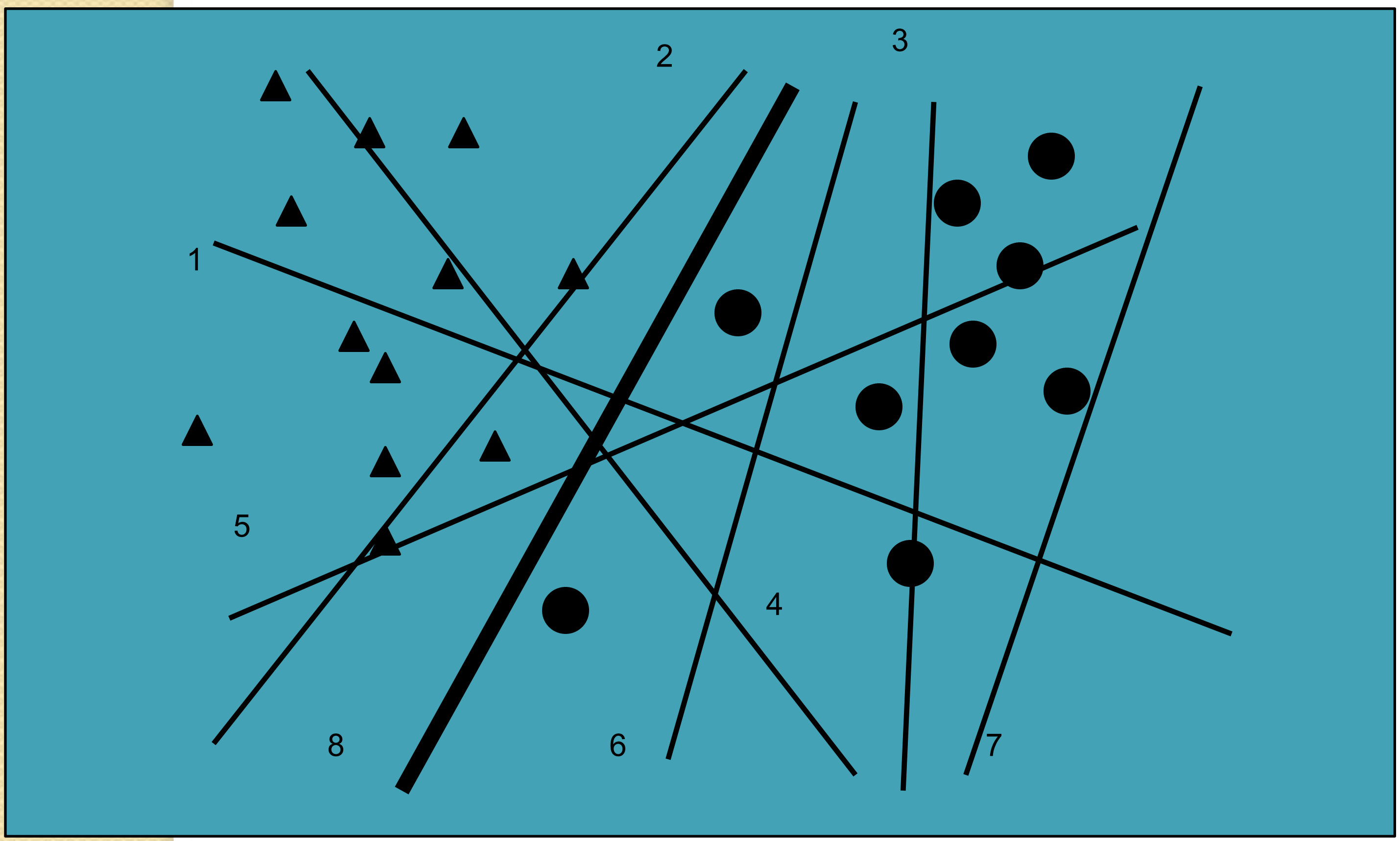
Características do Adaline

- Simples Operação
- Convergência Garantida

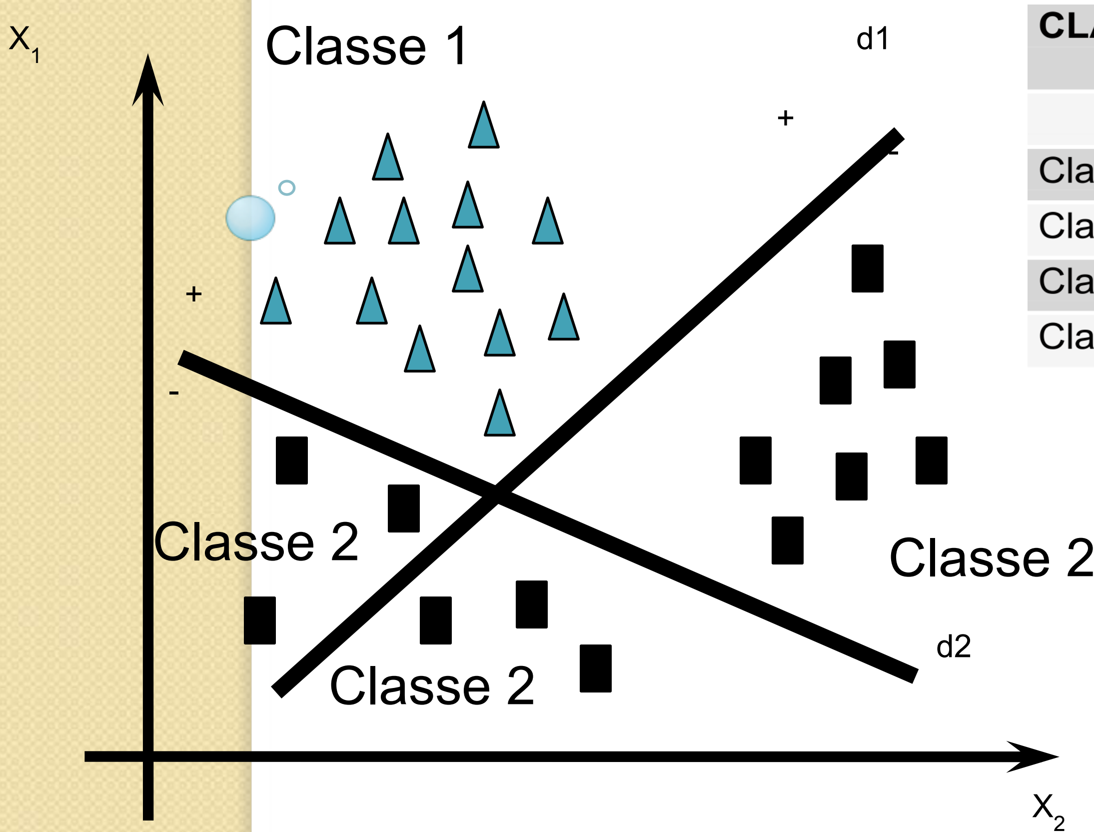


Capaz de resolver apenas problemas linearmente separáveis

Visualização do Treinamento

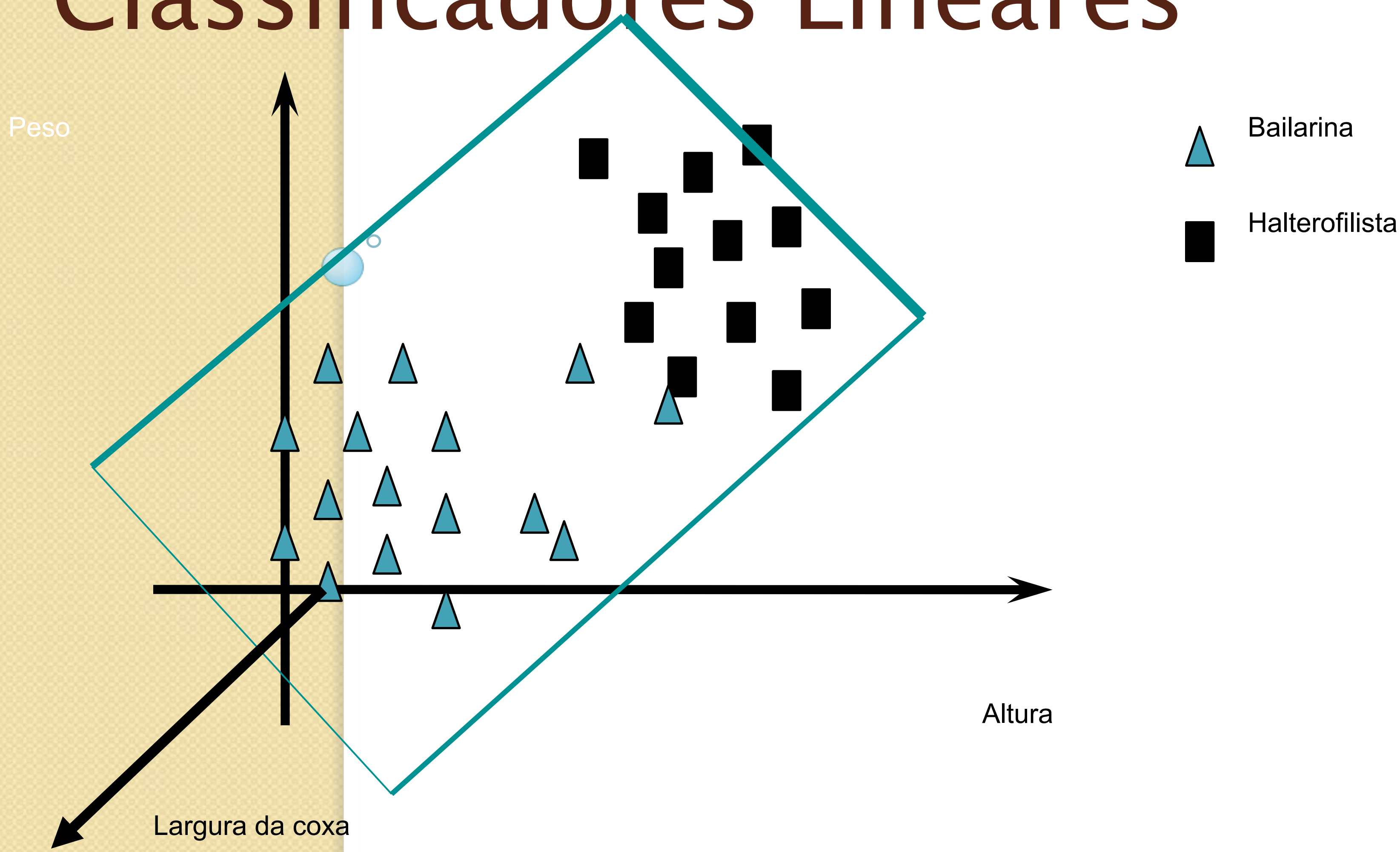


Classificadores lineares

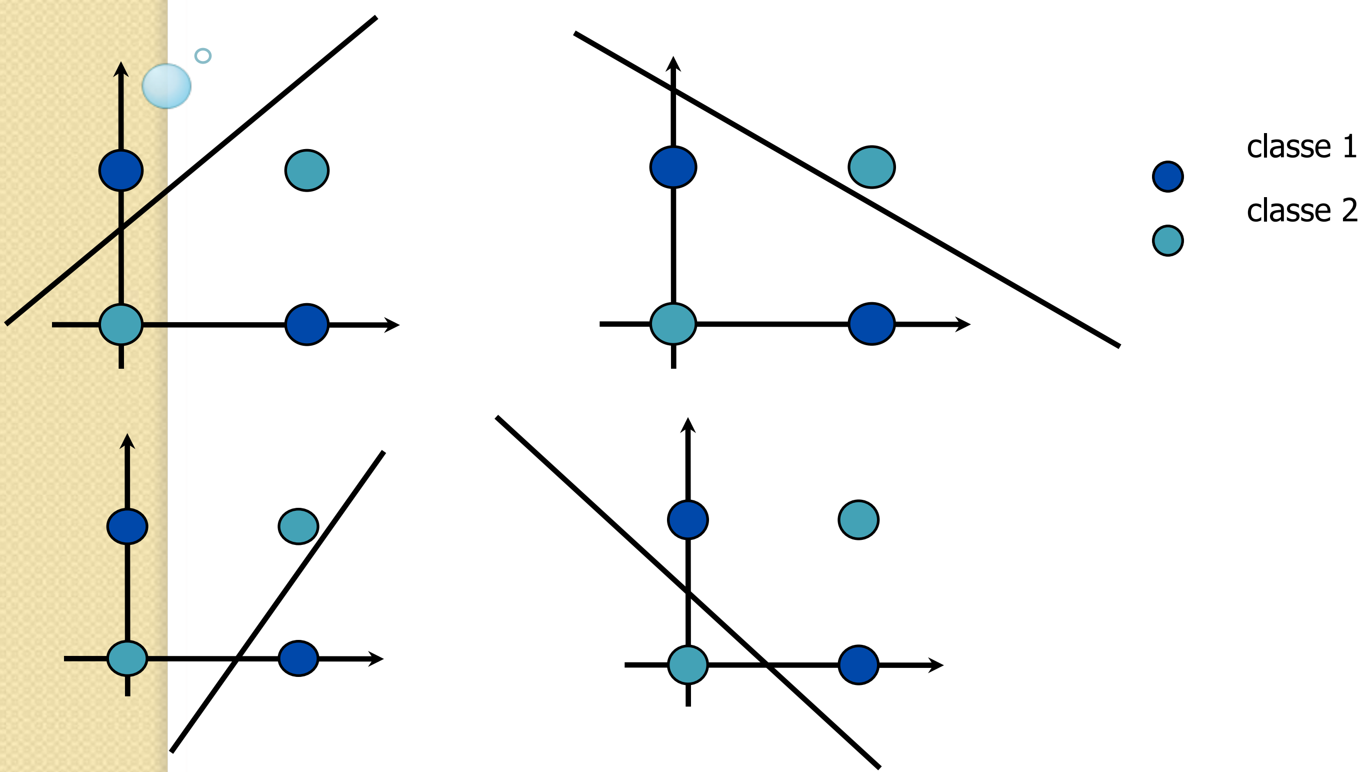


CLASSIFICAÇÃO	SINAL DA LINHA DE DECISÃO	
	d1	d2
Classe 1	+	+
Classe 2	+	-
Classe 2	-	+
Classe 2	-	-

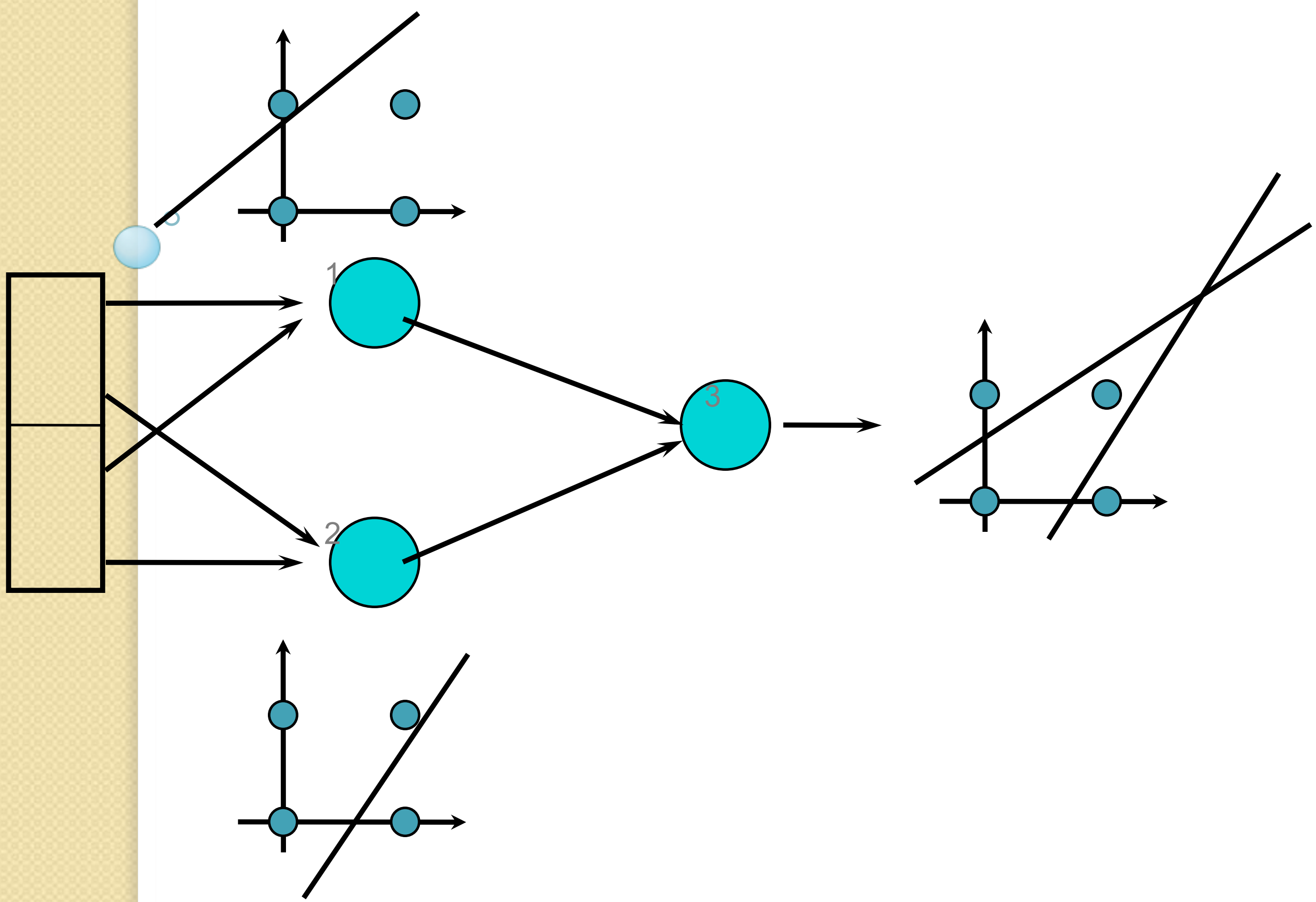
Classificadores Lineares



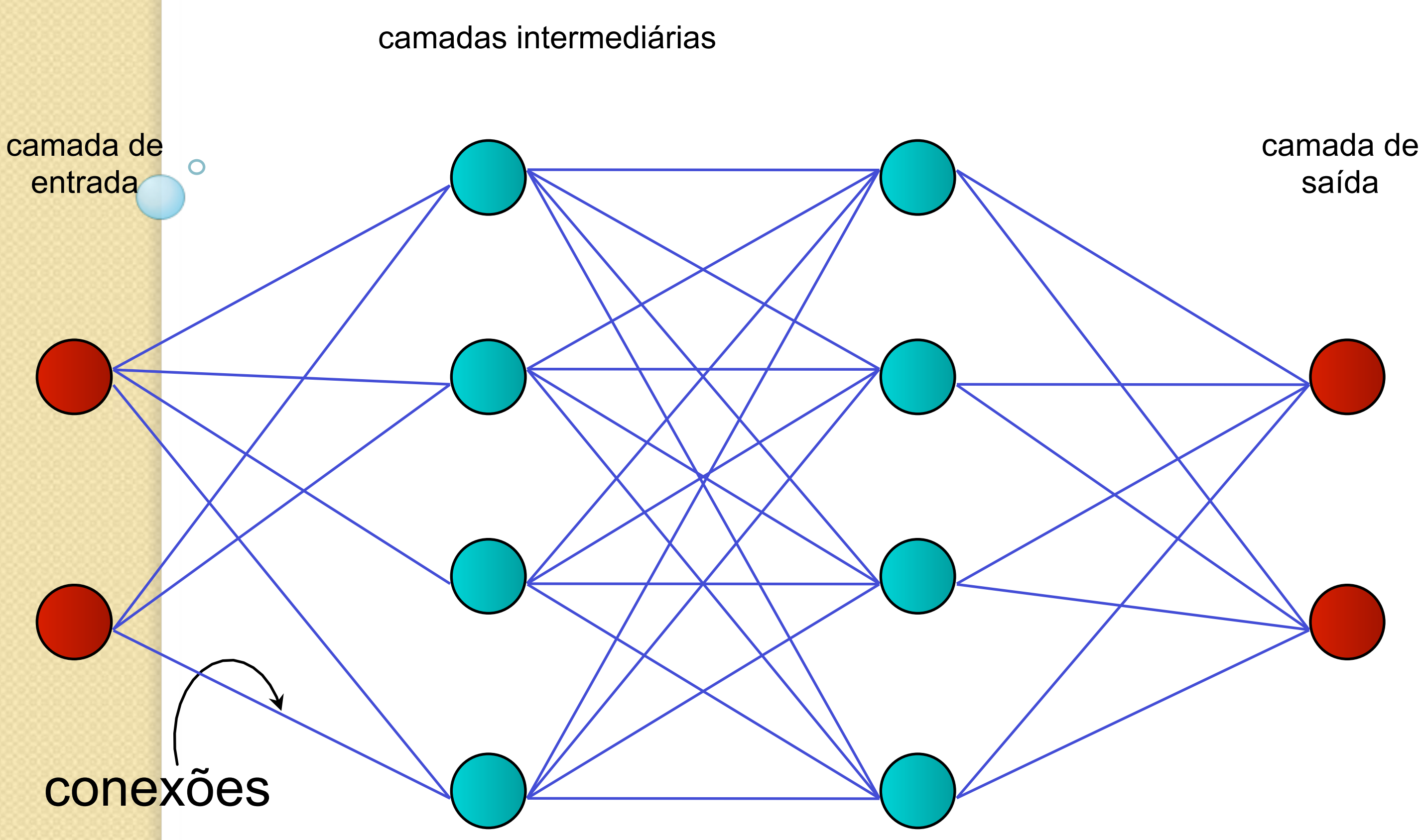
O Problema do Ou-exclusivo (XOR)



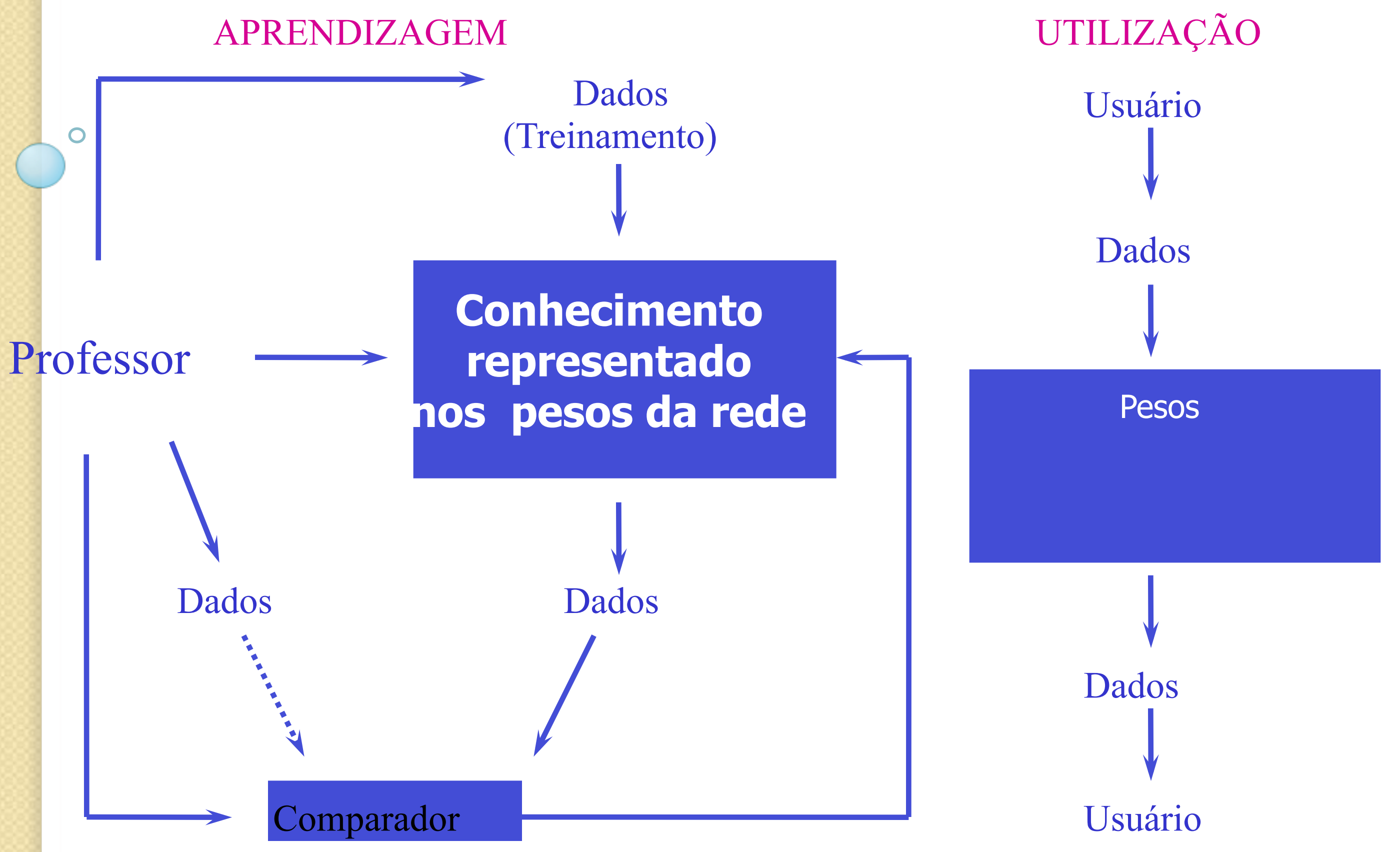
Solução para o XOR



Multilayer Perceptron (MLP) e Backpropagation (Regra Delta Generalizada)

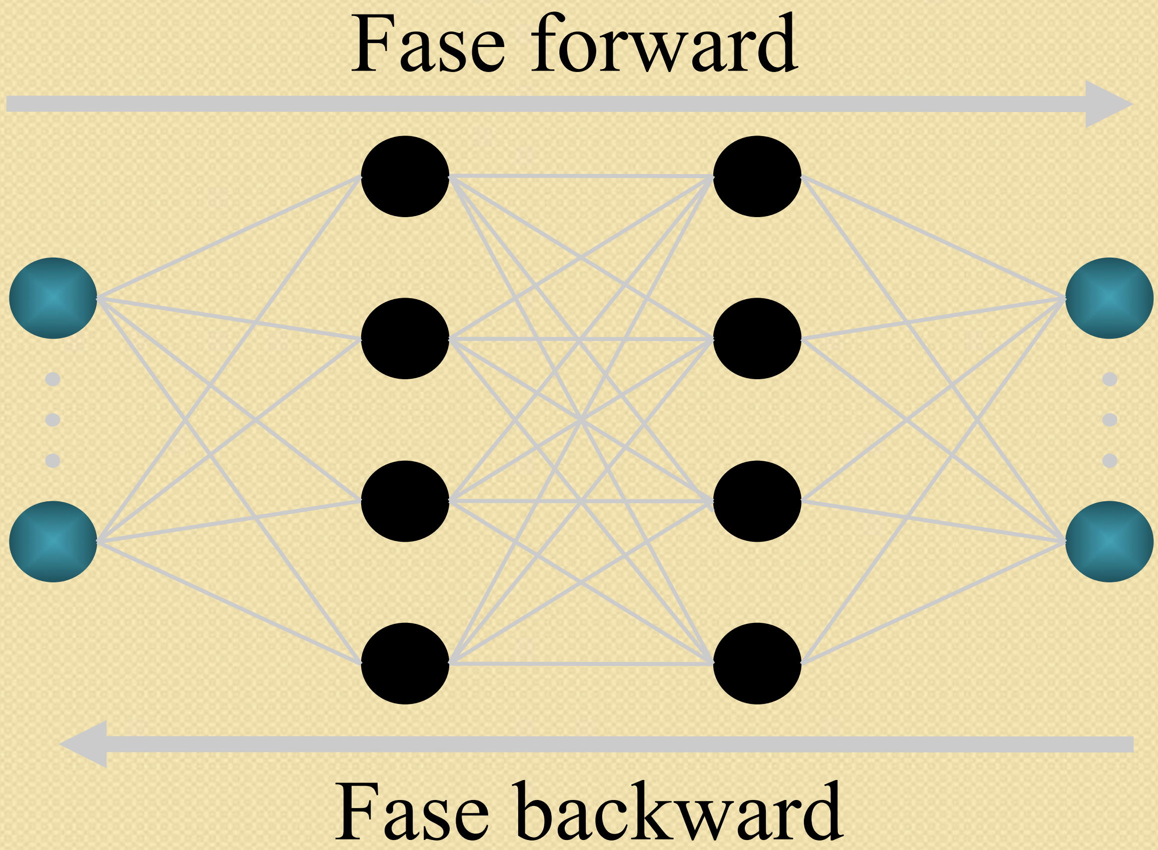


Funcionamento do MLP



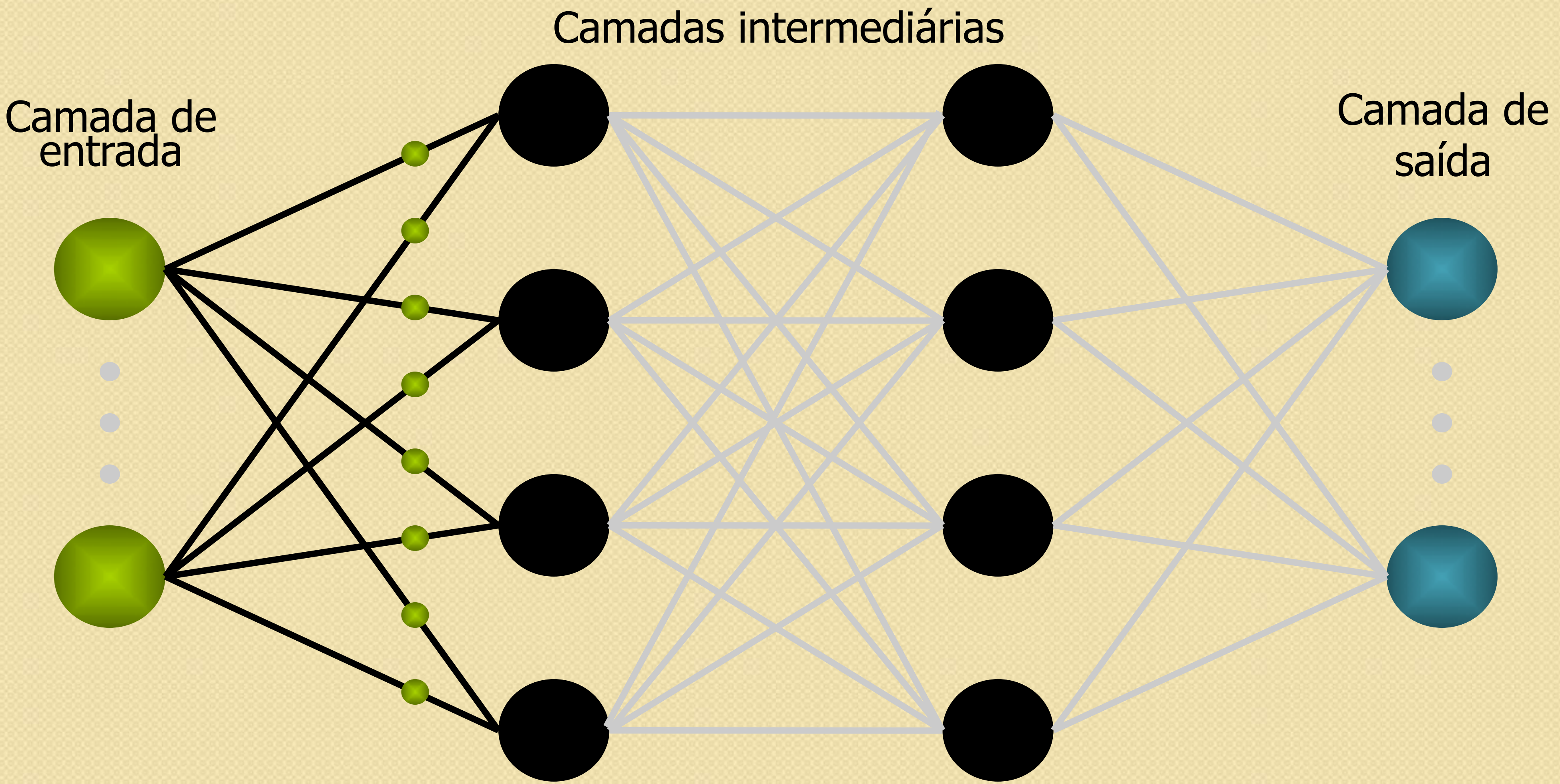
Algoritmo Backpropagation

- Treinamento em duas etapas:



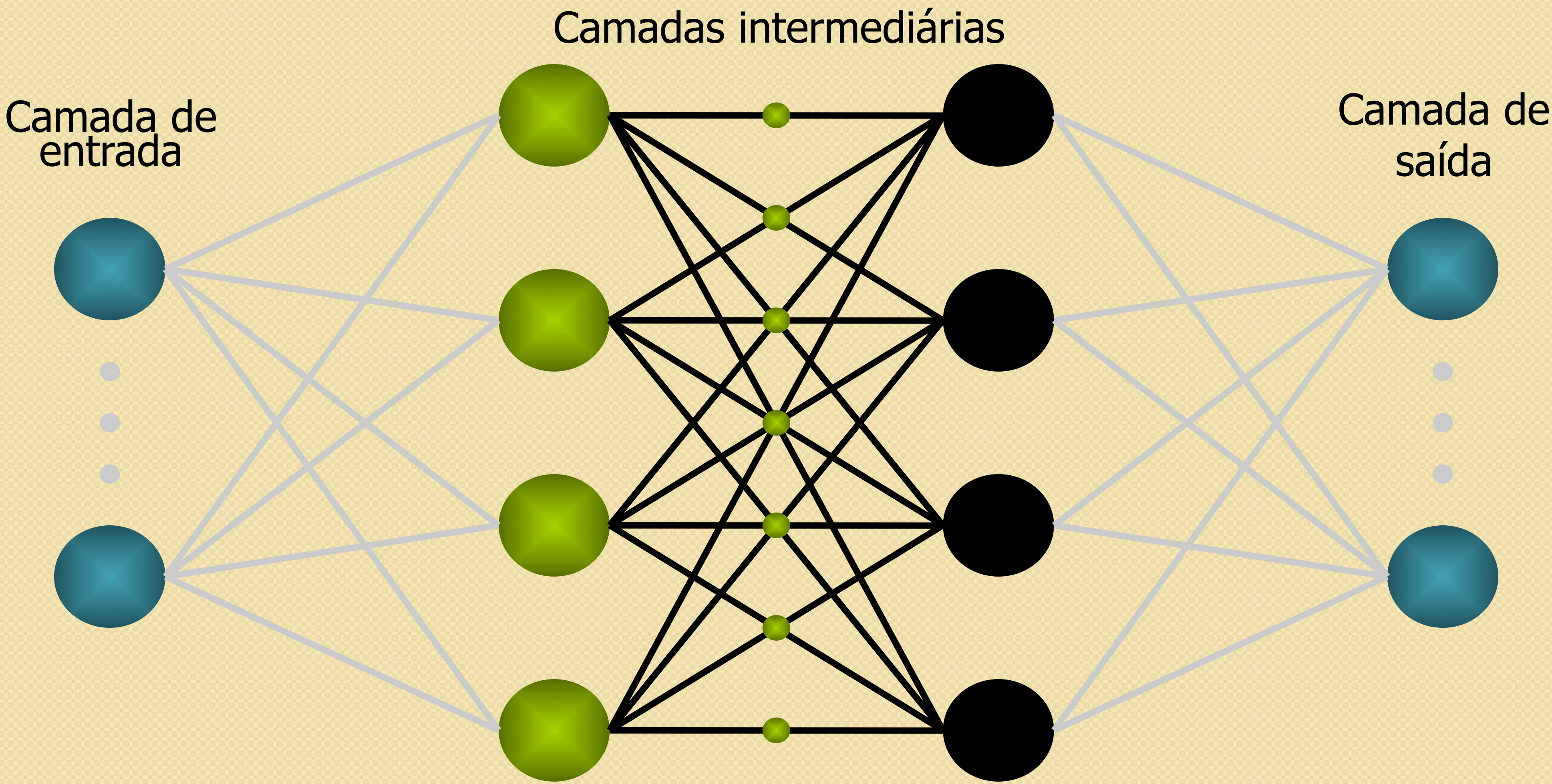
Fase forward

Entrada é apresentada à primeira camada da rede e propagado em direção às saídas



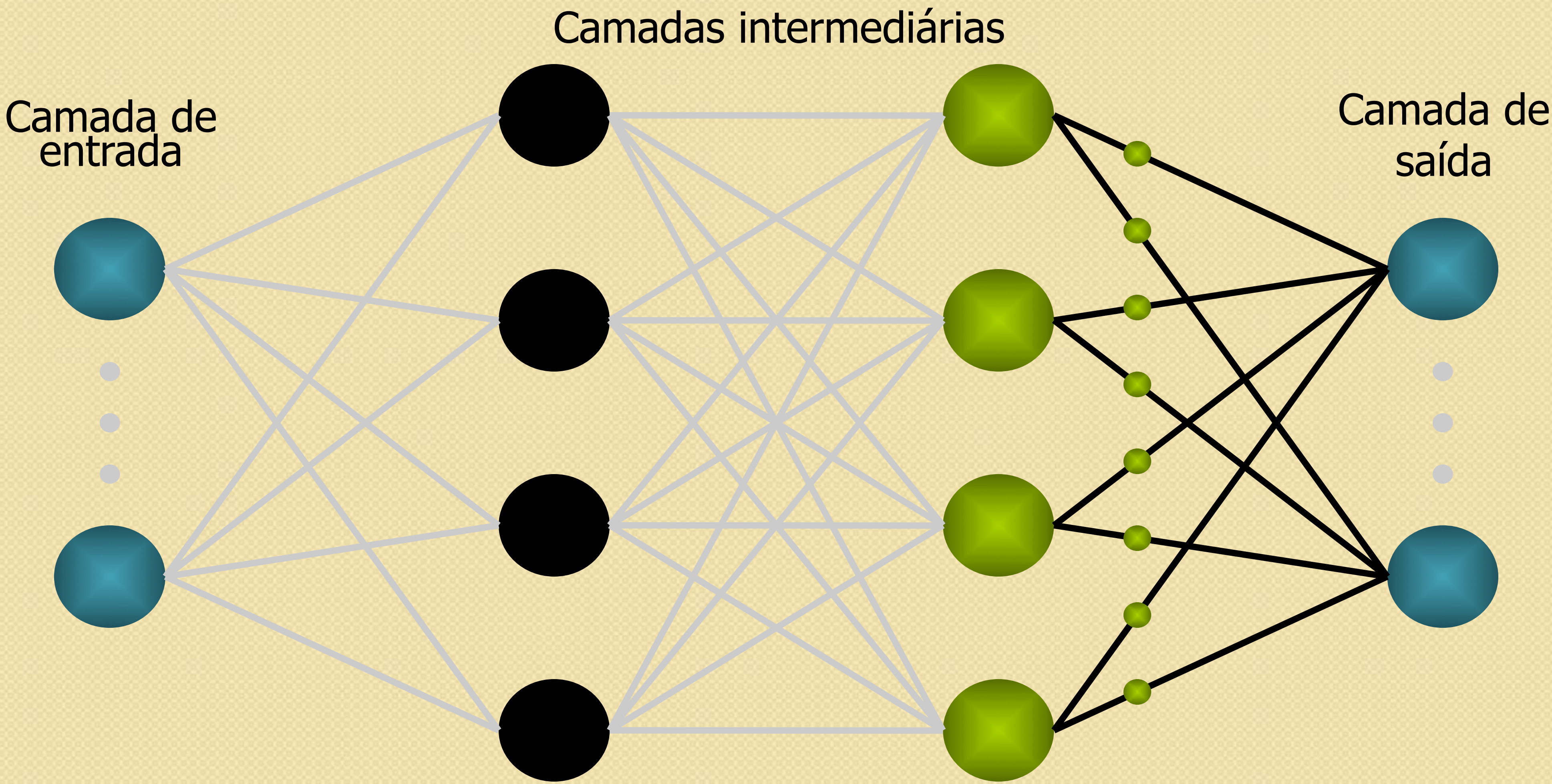
Fase forward

Os neurônios da camada **i** calculam seus sinais de saída e propagam à camada **i + 1**



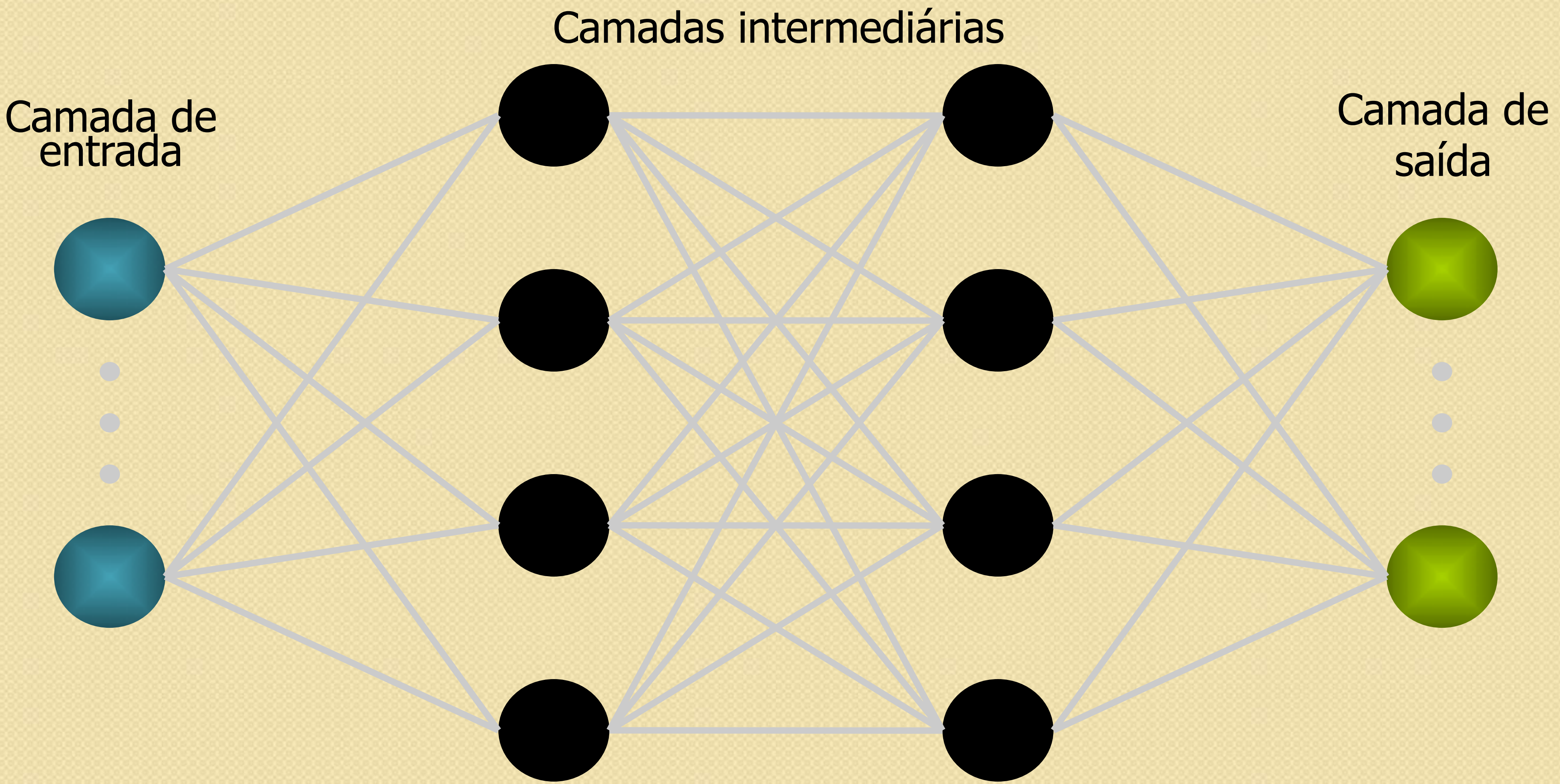
Fase forward

A última camada oculta calcula seus sinais de saída e os envia à camada de saída

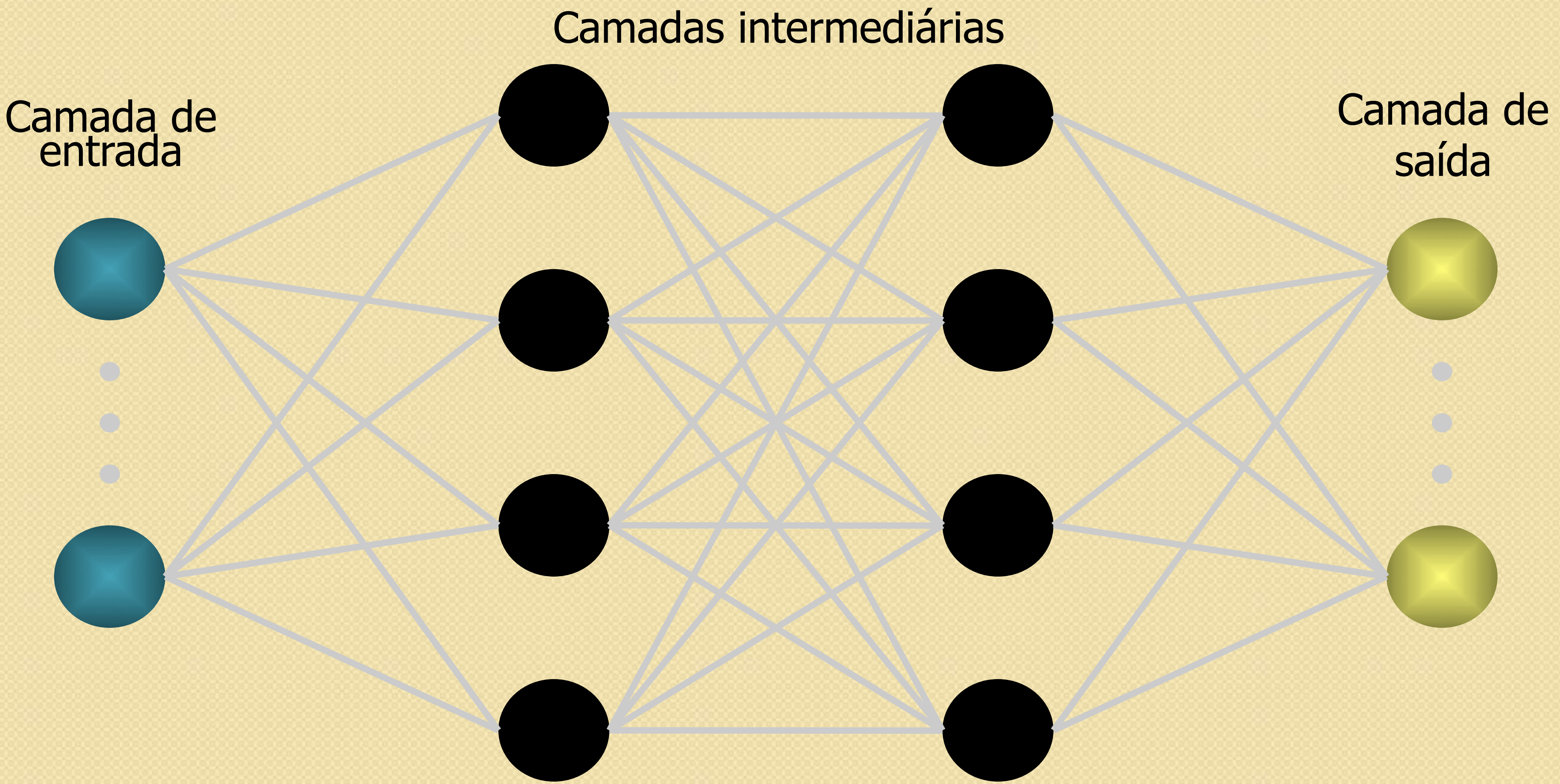


Fase forward

A camada de saída calcula os valores de saída da rede

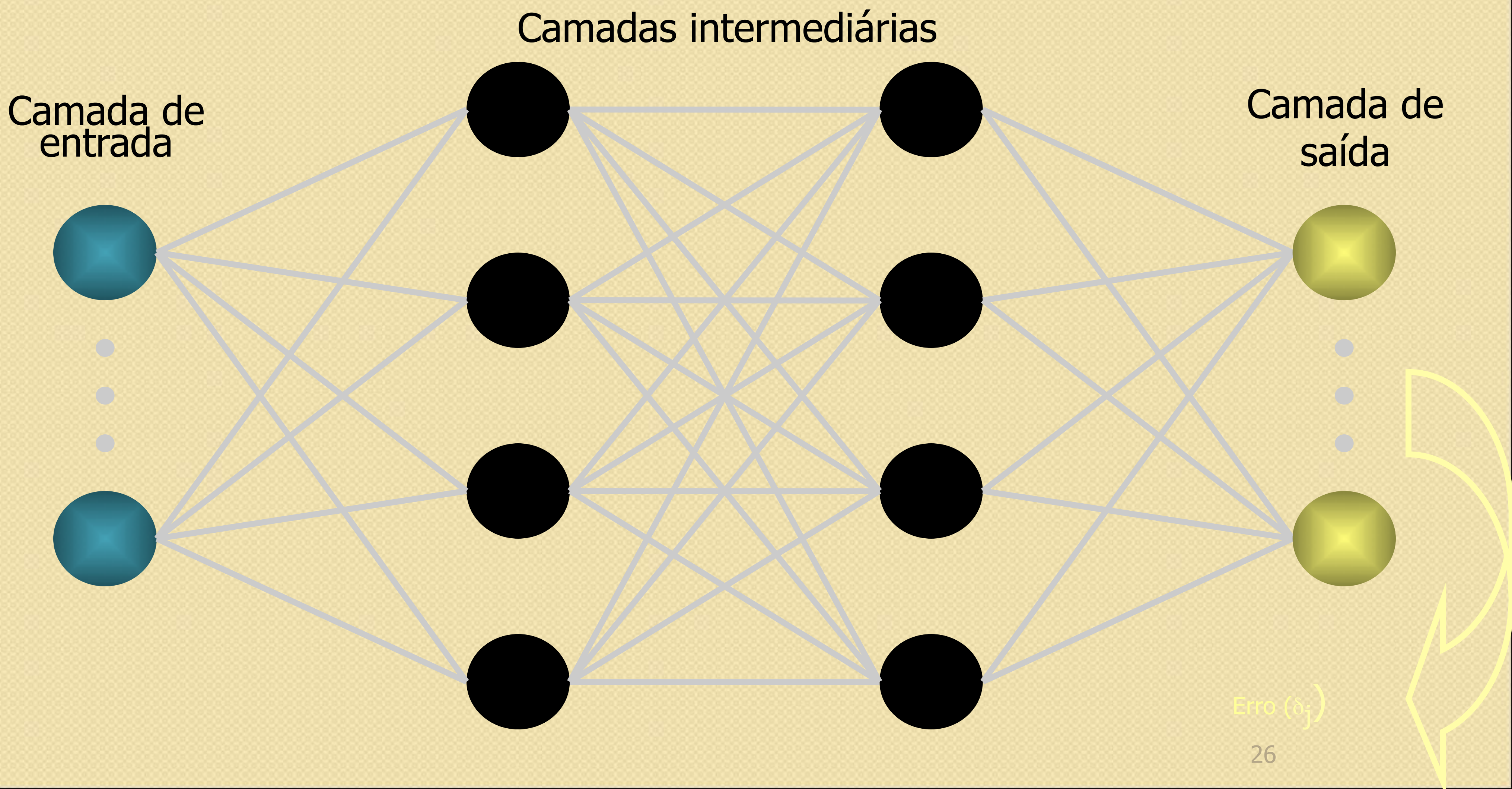


Fase backward



Fase backward

A camada de saída calcula o erro da rede: E_j

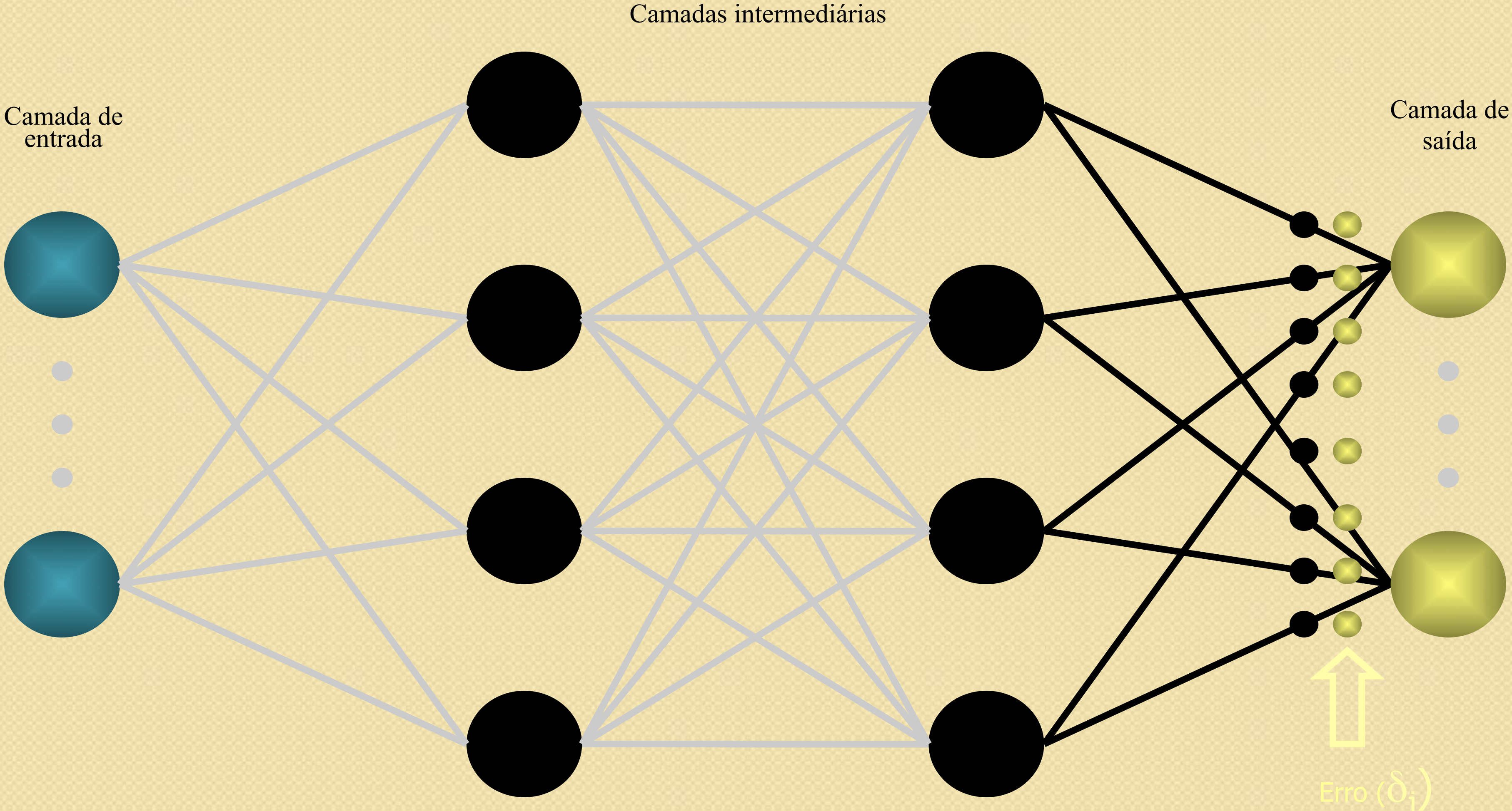


Calcula o termo de correção dos pesos

(a atualização será feita depois)

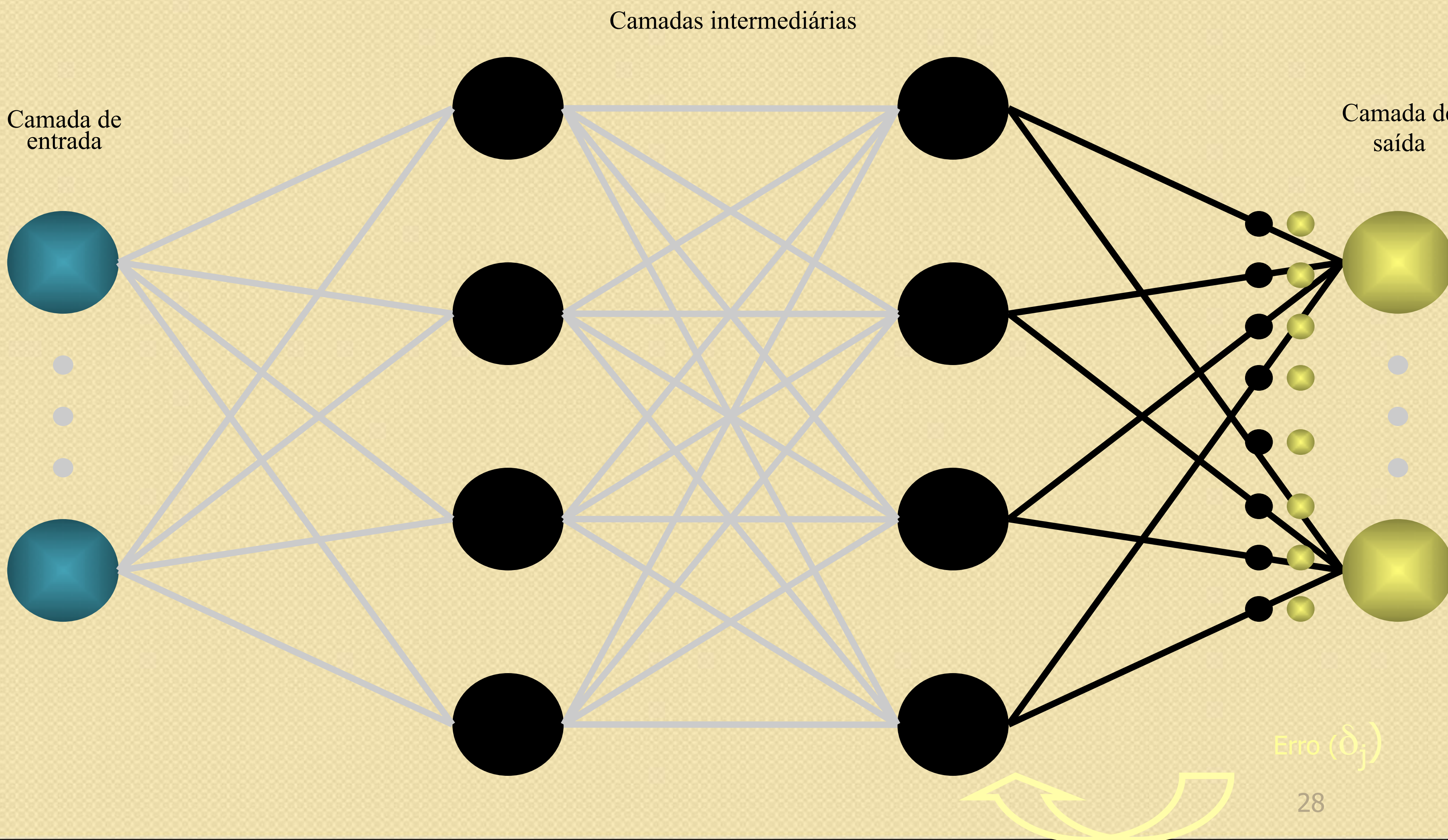
$\Delta w_{ji} = \alpha \delta_j x_i$

Fase backward



Fase backward

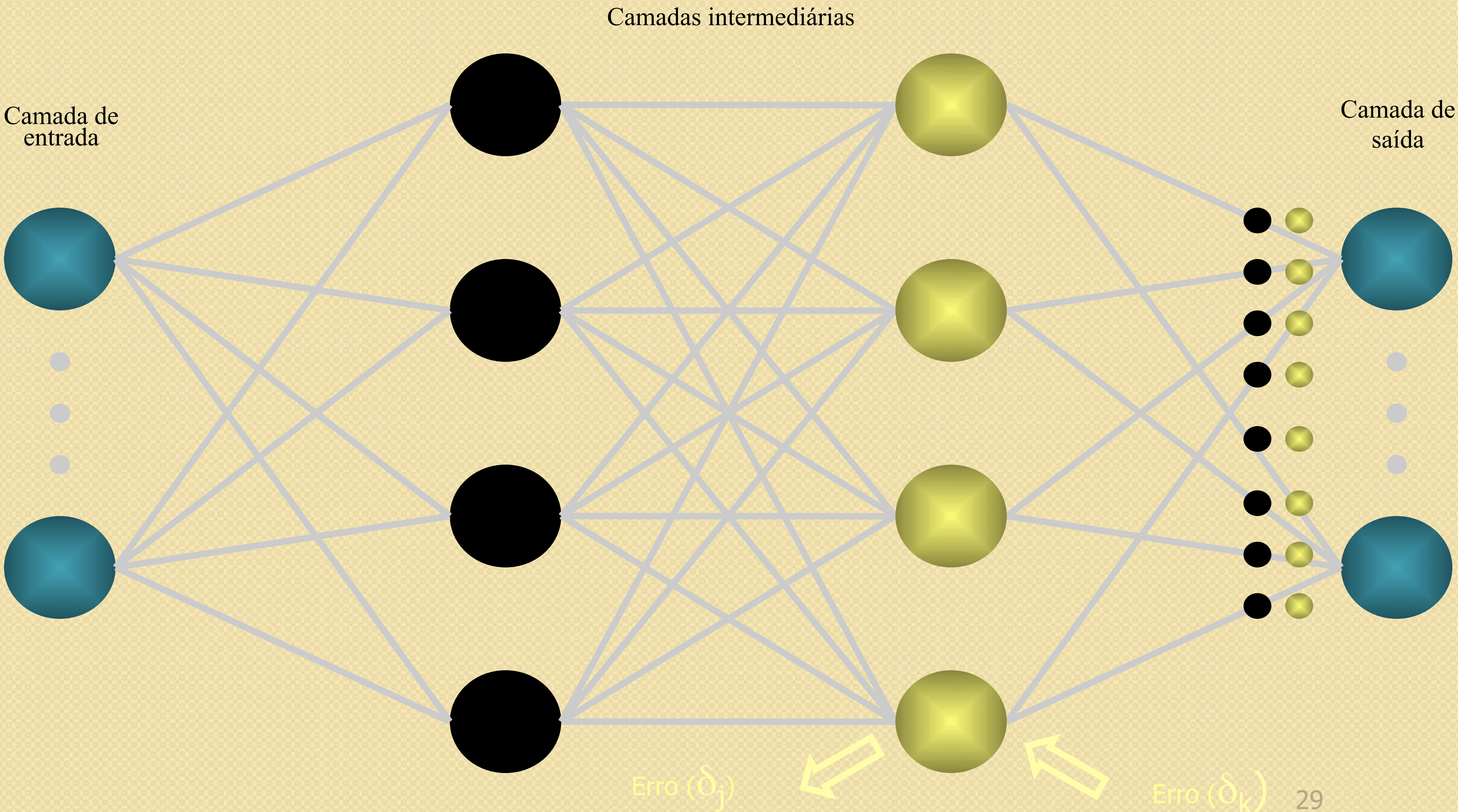
Envia o erro para a
última camada oculta



A camada oculta calcula o seu erro

$\delta_j = f'(u_j). \sum \delta_k w_{lk}$

Fase backward

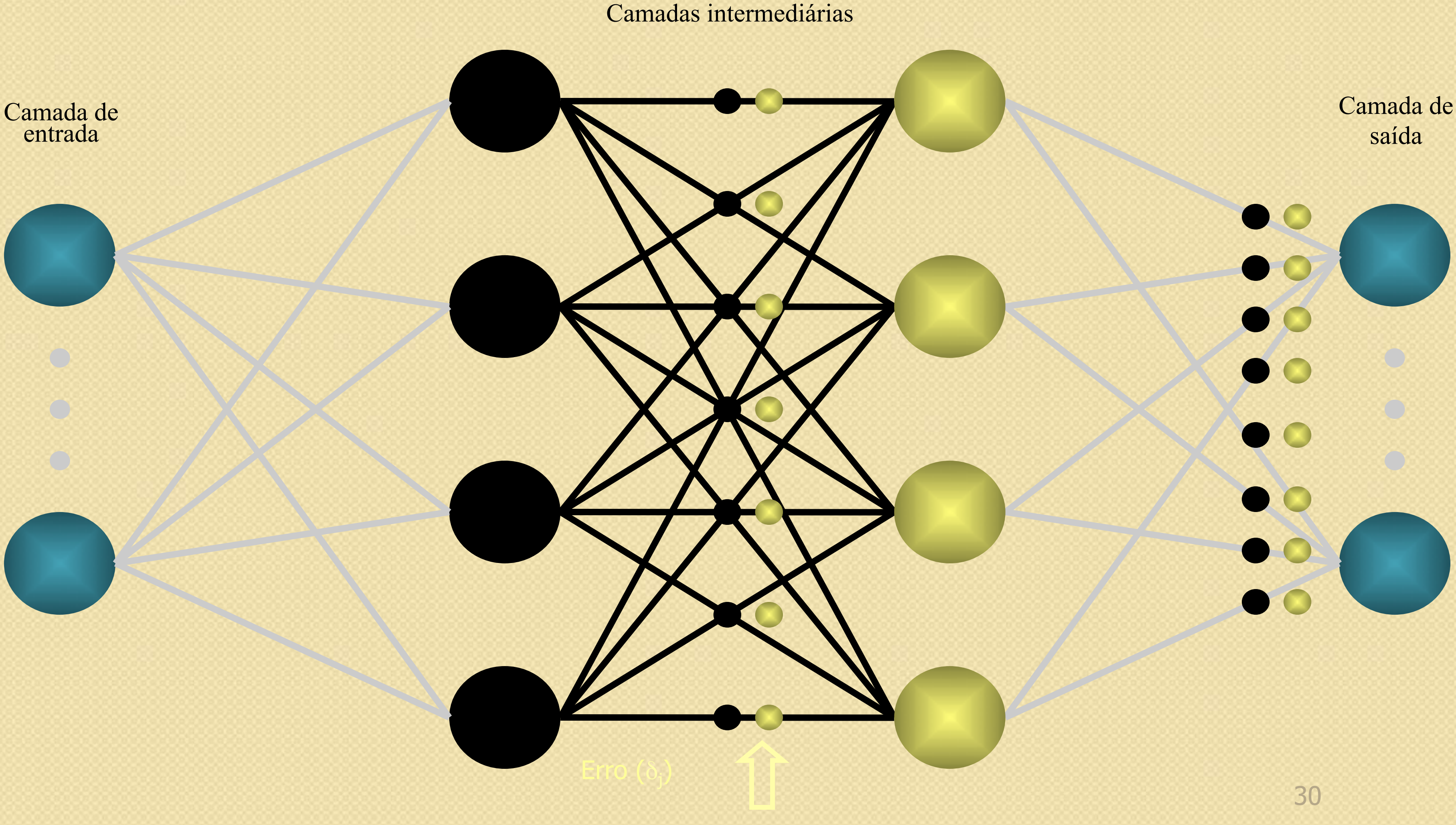


Calcula o termo de correção dos pesos

(a atualização será feita depois)

$\Delta w_{ij} = \alpha \delta_j x_i$

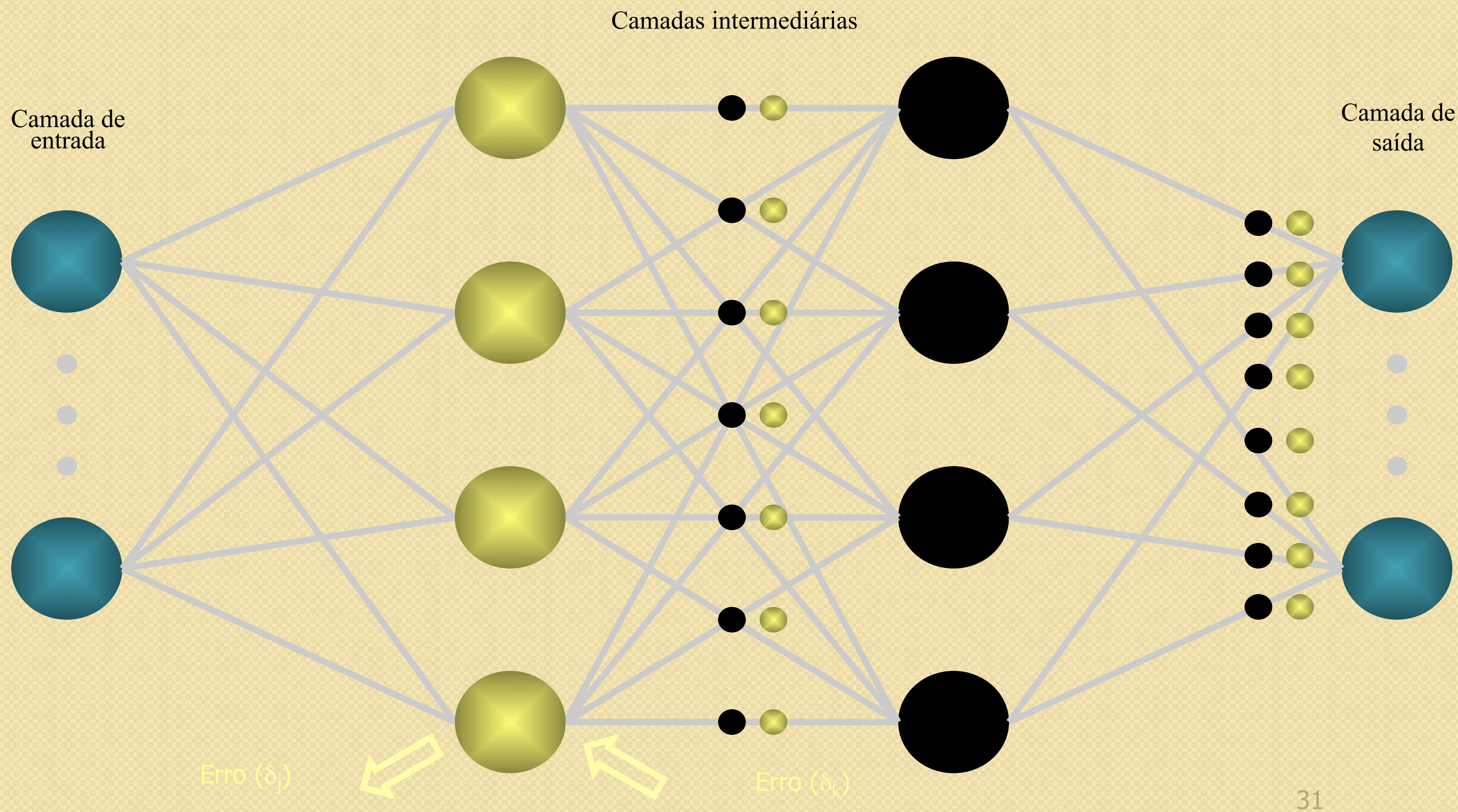
Fase backward



Fase backward

A camada oculta calcula o seu erro

$\delta_j = f'(u_j).\sum \delta_k w_{lk}$

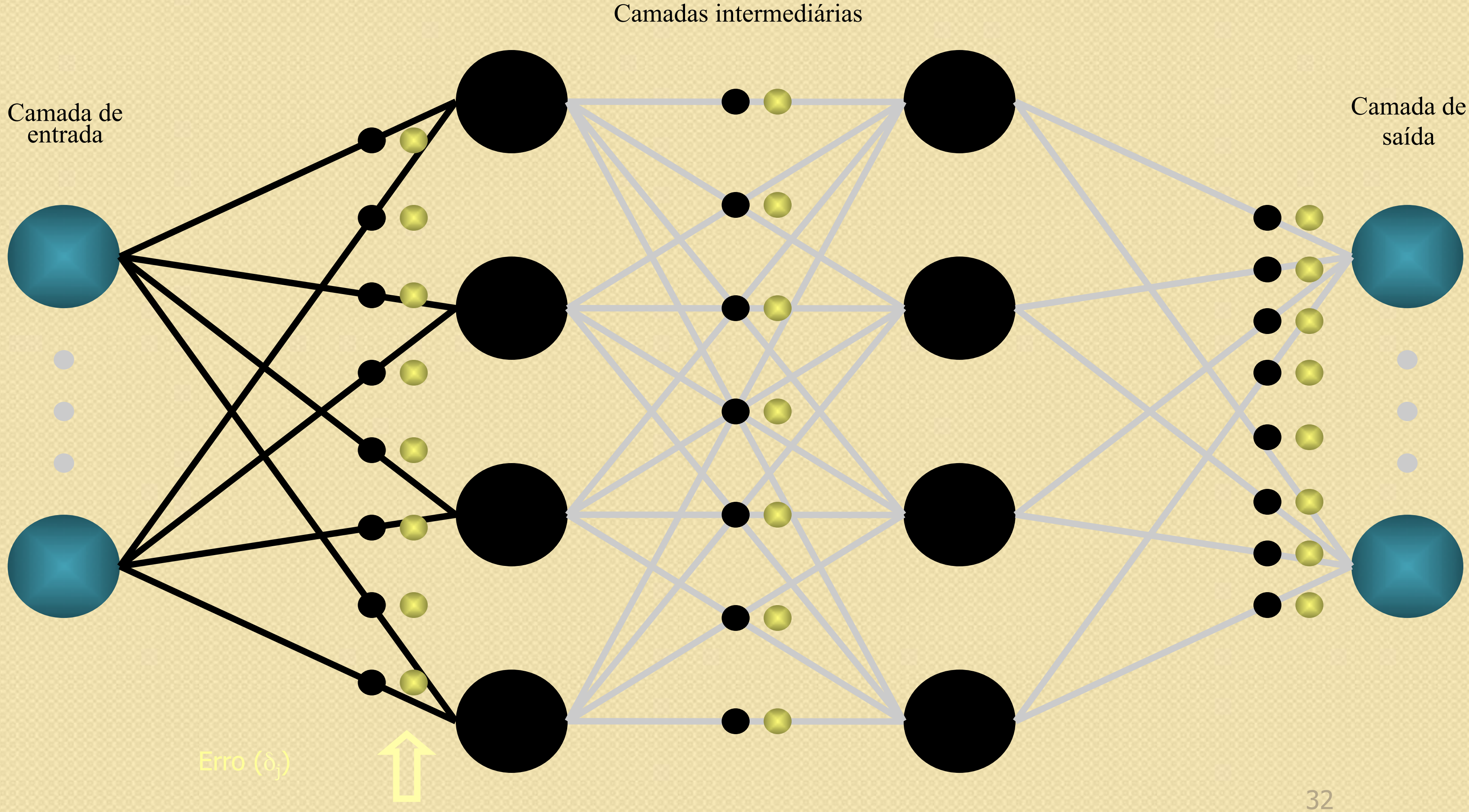


Calcula o termo de correção dos pesos

(a atualização será feita depois)

$\Delta w_{ij} = \alpha \delta_j x_i$

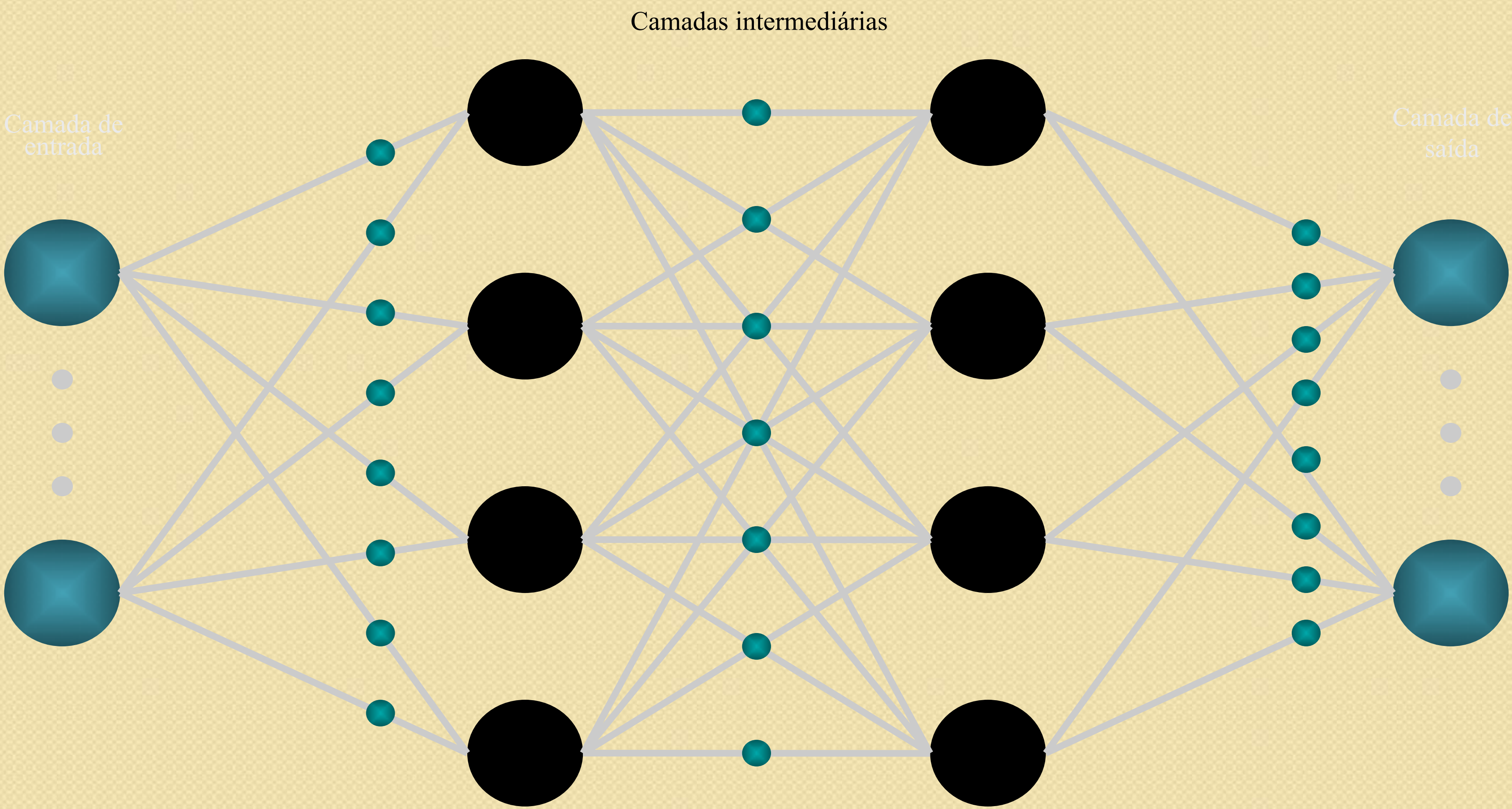
Fase backward



Cada unidade atualiza seus pesos

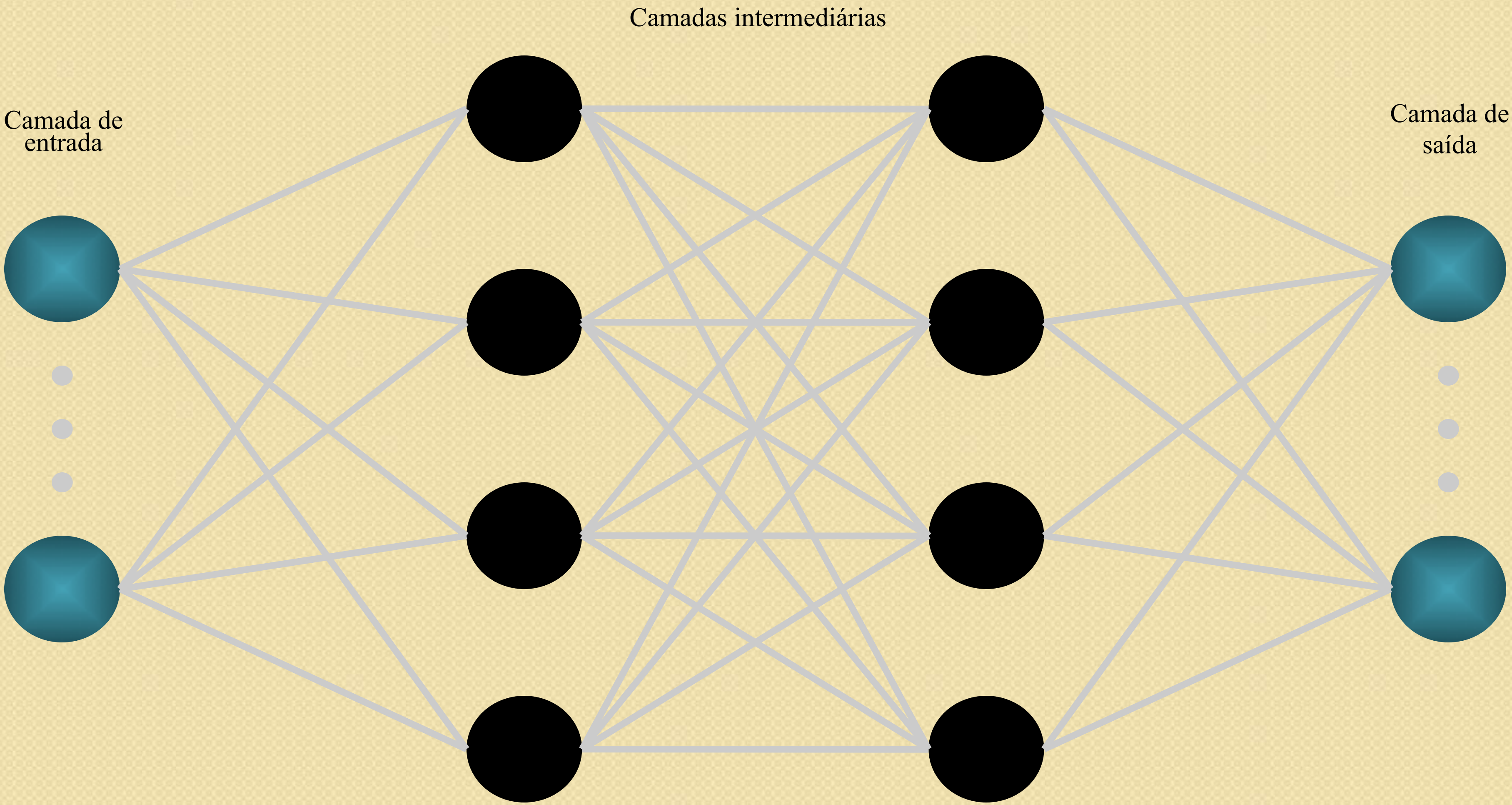
$w_{ij}(novo) = w_{ij}(velho) + \Delta w_{jk}$

Fase backward



Repete-se o processo enquanto
enquanto a rede não aprender
o padrão de entrada

Error-Backpropagation



Funcionamento do MLP

- Duas fases de operação
  - Passo para frente (forward pass)
  - Passo para trás (backward pass)

- Dado um conjunto de pares (X_p, Y_p), construir um mapeamento $F(W; X_p) => Y_p$

Como construir $F(W; X_p) => Y_p$?

Regra Delta Generalizada ou Error-Back Propagation

O erro na camada de saída:

$$E_p = \frac{1}{2}\sum_{j=1}^{n}(t_{pj} - o_{pj})^2$$

Para minimizar o erro :

$$\Delta_p W_{ji} \propto -\frac{\partial E_p}{\partial W_{ji}}$$

Regra Delta Generalizada ou Error-Back Propagation

- $\Delta_p W_{ji} \propto \delta_j \,.\, O_{pi}$

- $\Delta_p W_{ji} = \eta \,.\, \delta_j \,.\, O_{pi}$

Regra Delta Generalizada ou Error-Back Propagation

- Dois casos precisam ser considerados para δ_{pj}

$$\frac{\partial E_p}{\partial W_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial W_{ji}}$$

$$\frac{\partial net_{pj}}{\partial W_{ji}} = \frac{\partial}{\partial W_{ji}} \sum_k W_{jk} O_{pk} = O_{pi}$$

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \Rightarrow -\frac{\partial E_p}{\partial W_{ji}} = \delta_{pj} O_{pi}$$

$$\Delta_p W_{ji} = \eta \delta_{pj} . O_{pi}$$

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial O_{pj}} \frac{\partial O_{pj}}{\partial net_p}$$

$$\frac{\partial O_{pj}}{\partial net_{pj}} = f_j^{|}(net_{pj})$$

$$\frac{\partial E_p}{\partial O_{pj}}$$

(2 casos precisam ser considerados!)

Primeiro caso : j é uma unidade de saída

$$\frac{\partial E_p}{\partial O_{pj}} = -(t_{pj} - o_{pj})$$

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j^{|}(net_{pj})$$

Segundo caso : j é uma unidade intermediária

$$\frac{\partial E_p}{\partial O_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}}$$

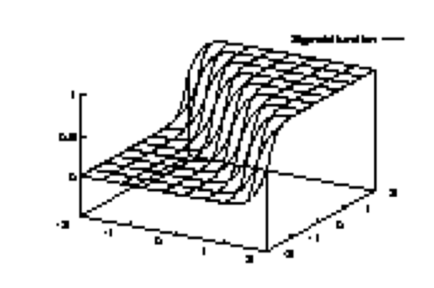
$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial O_{pj}} \sum_i w_{ik} o_{pi}$$

$$\Rightarrow \frac{\partial E_p}{\partial o_{pj}} = - \sum \delta_{pk} w_{jk}$$

$$\delta_{pj} = f_j^{\prime}(net_{pj}) \sum \delta_{pk} w_{jk}$$

E a função de ativação f ?

Considerando uma função sigmoid



$$f'(net_{pj})?$$

$$o_{pj} = f(net_{pj}) = \frac{1}{1+\exp(-net_{pj})}$$

$$f'(net_{pj}) = o_{pj}(1-o_{pj})$$

Características do MLP

- Aproximador Universal de Funções
  - Uma única camada intermediária é capaz de aproximar qualquer função contínua definida em um hipercubo
- Alta capacidade de generalização
- Convergência para mínimo global não garantida
- Em alguns casos, lento na aprendizagem

Outra Função Erro: Entropia Cruzada (Cross-Entropy)

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)],$$

De uma maneira geral...

Uma rede neural pode ser vista como um conjunto de funções $Y_k(X_p;W)$, tal que dado $X_p \Rightarrow Y_p$

No caso de classificação

$Y_k = 1$ se $X_p \in k$

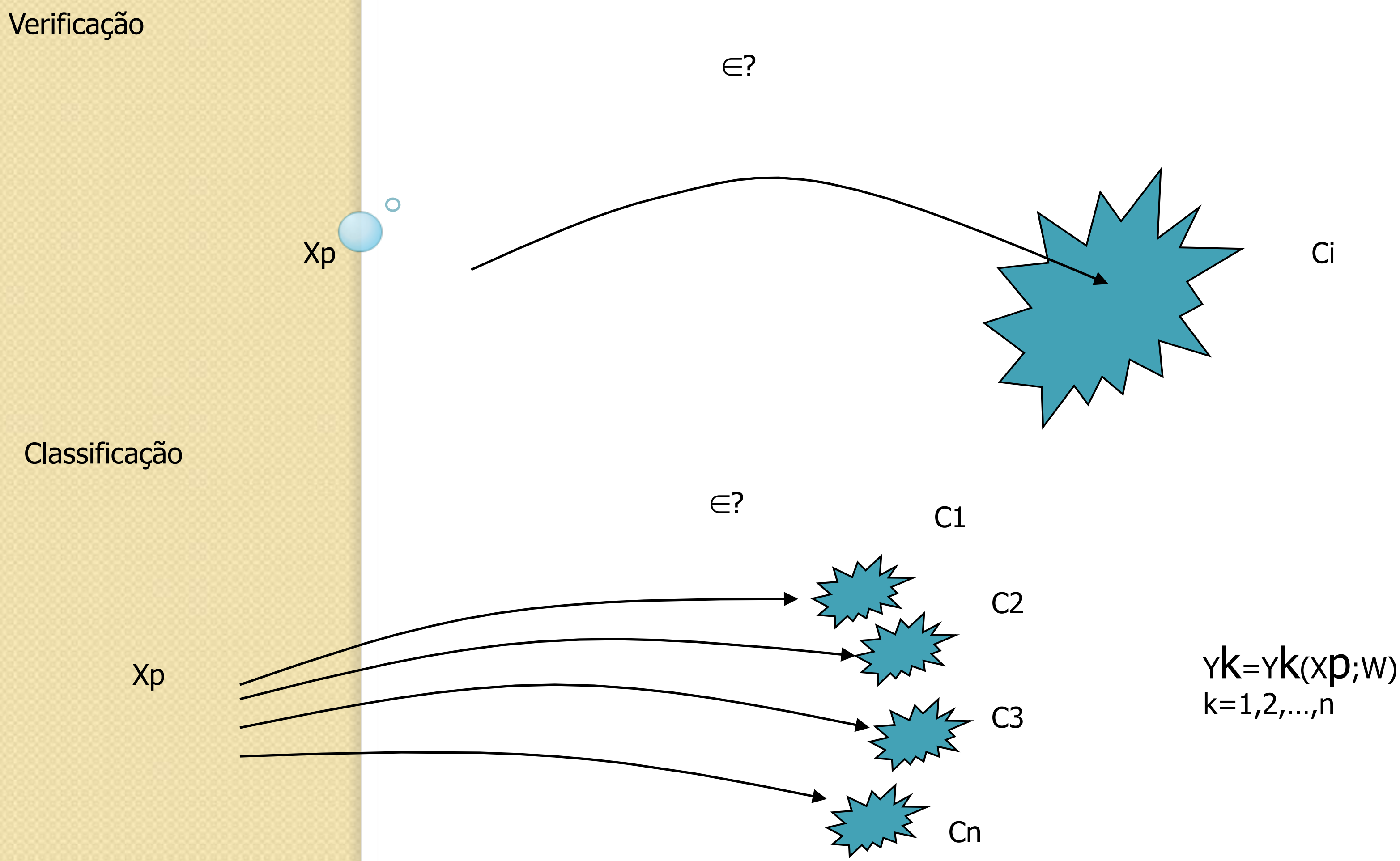
0, caso contrário

No caso em que Y_k são variáveis contínuas

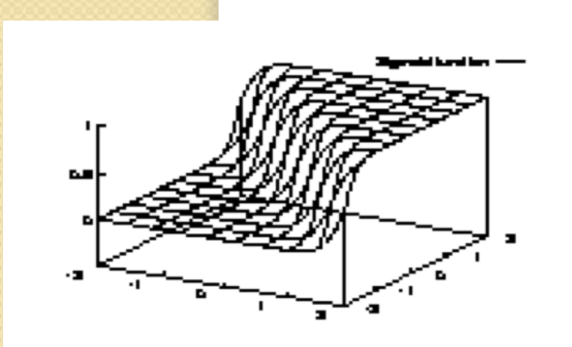
⇒ problema de regressão

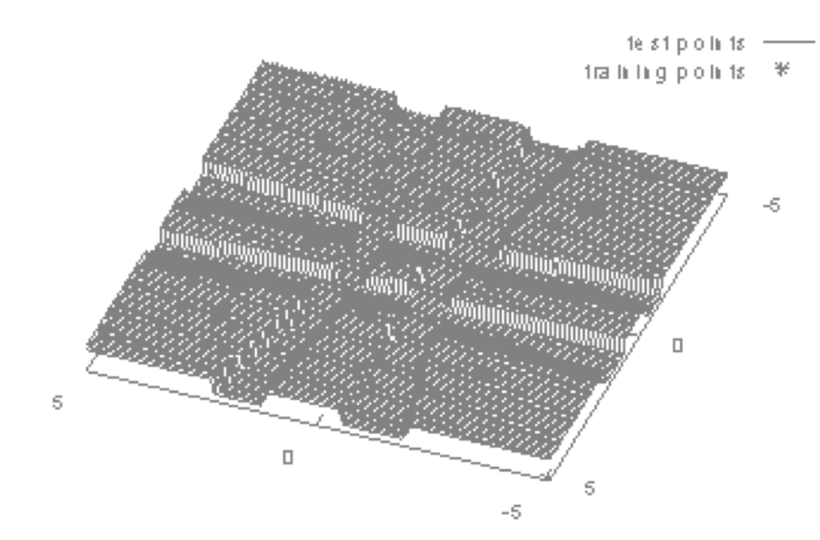
⇒ ou problema de aproximação de funções

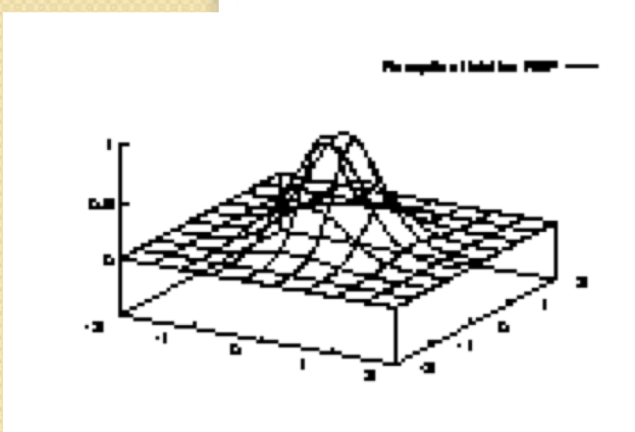
Reconhecimento de Padrões

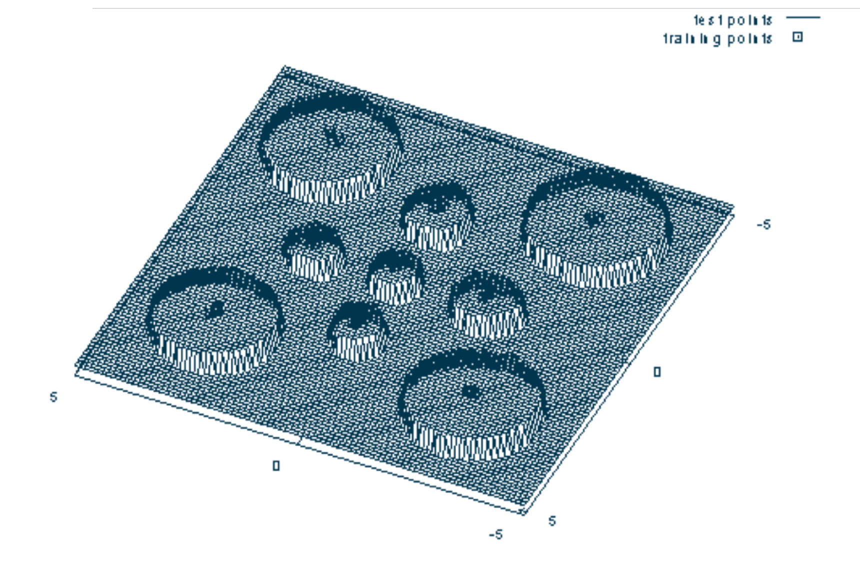


Reconhecimento de Padrões

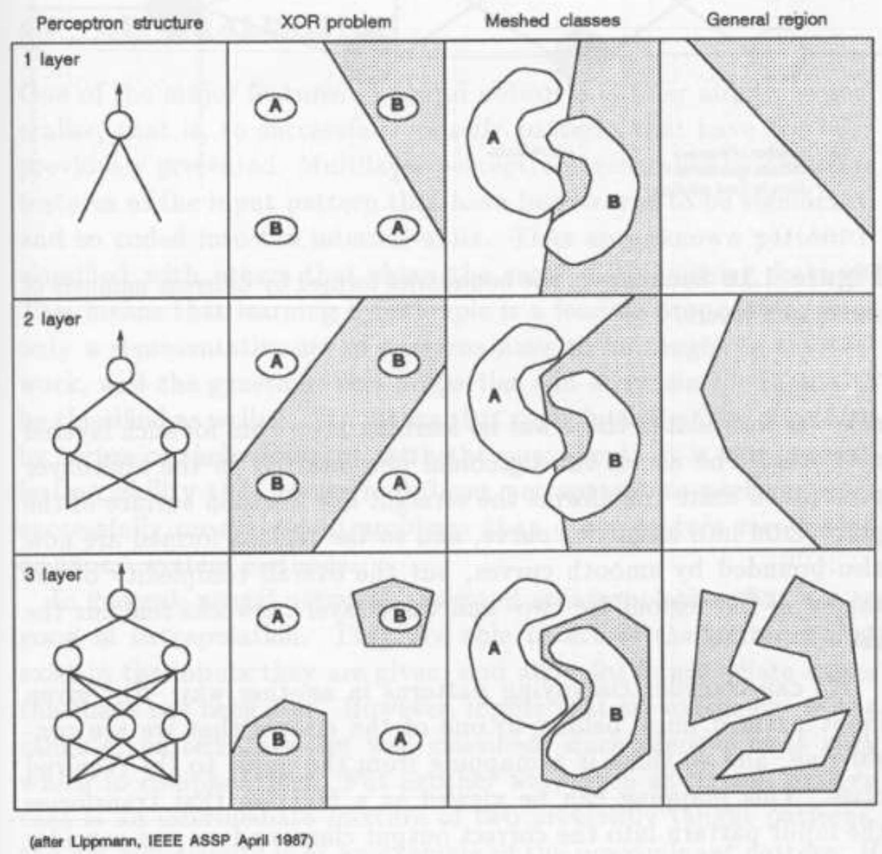




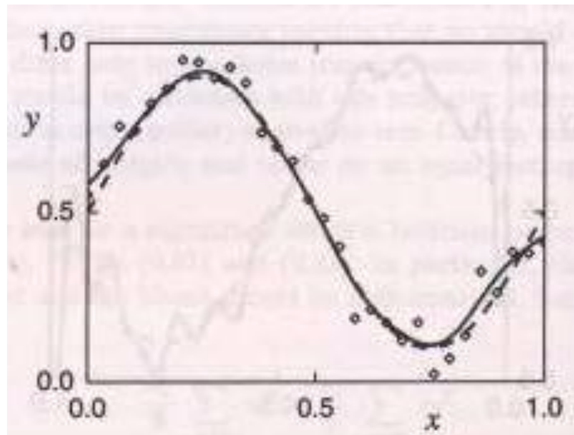


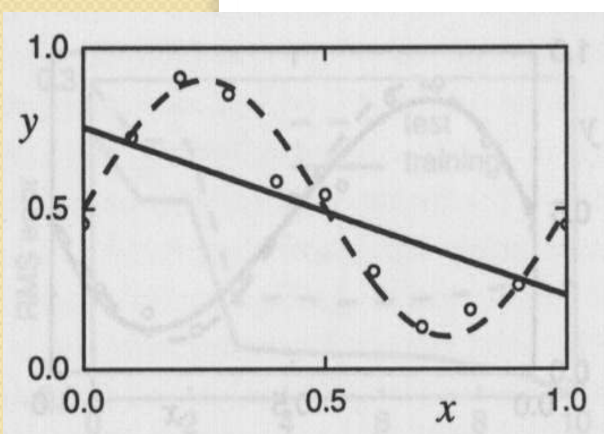


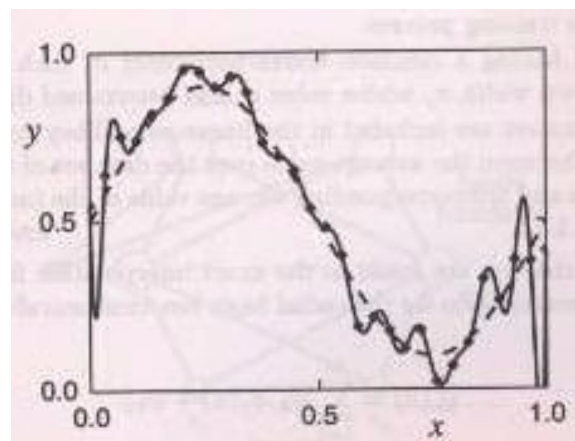
Complexidade Funcional do MLP x Número de Camadas



Complexidade Funcional versus Over-fitting







Treinamento com Validação Cruzada

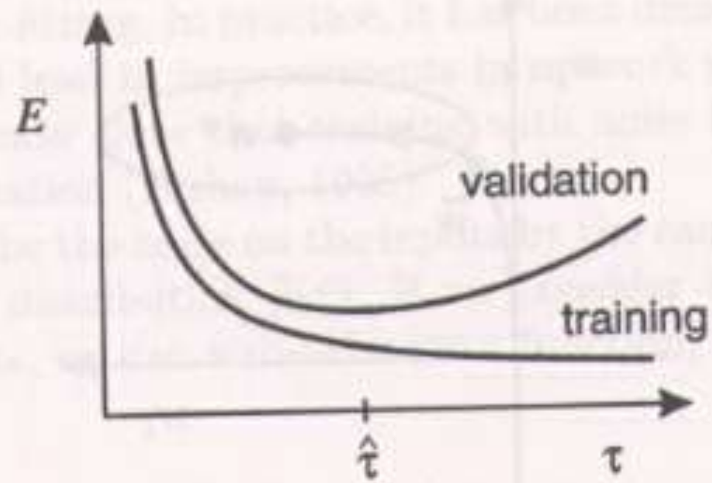
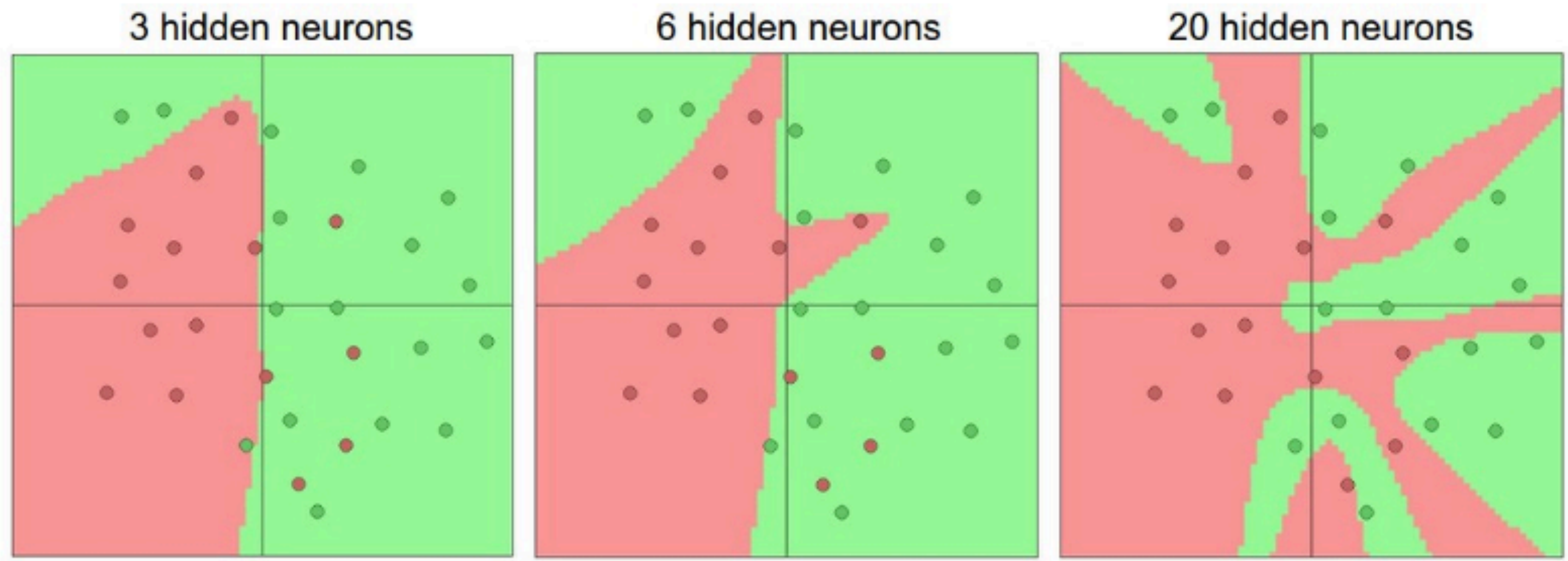


Figure 9.7. A schematic illustration of the behaviour of training and validation set errors during a typical training session, as a function of the iteration step τ. The goal of achieving the best generalization performance suggests that training should be stopped at the point $\hat{\tau}$ corresponding to the minimum of the validation set error.

Ainda Sobre Overfitting



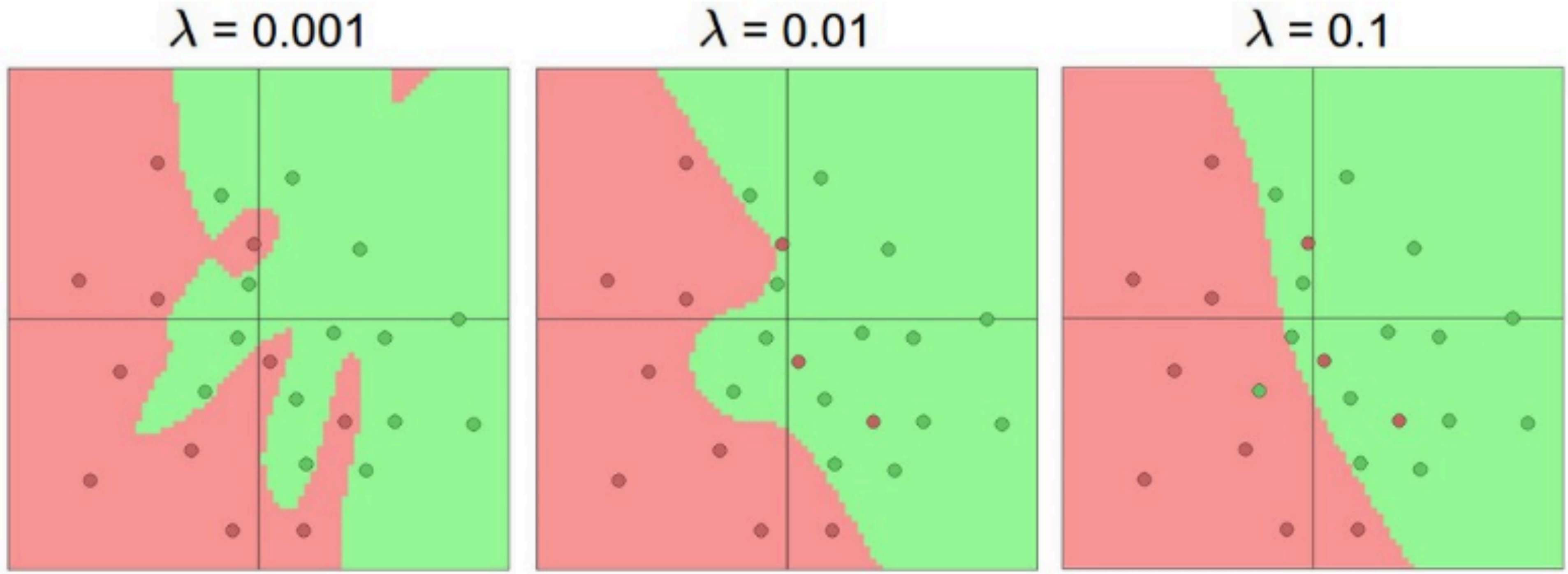
Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.

Regularização (Regularization)

$$C = \frac{1}{2n} \sum_x \|y - a^L\|^2 + \frac{\lambda}{2n} \sum_w w^2.$$

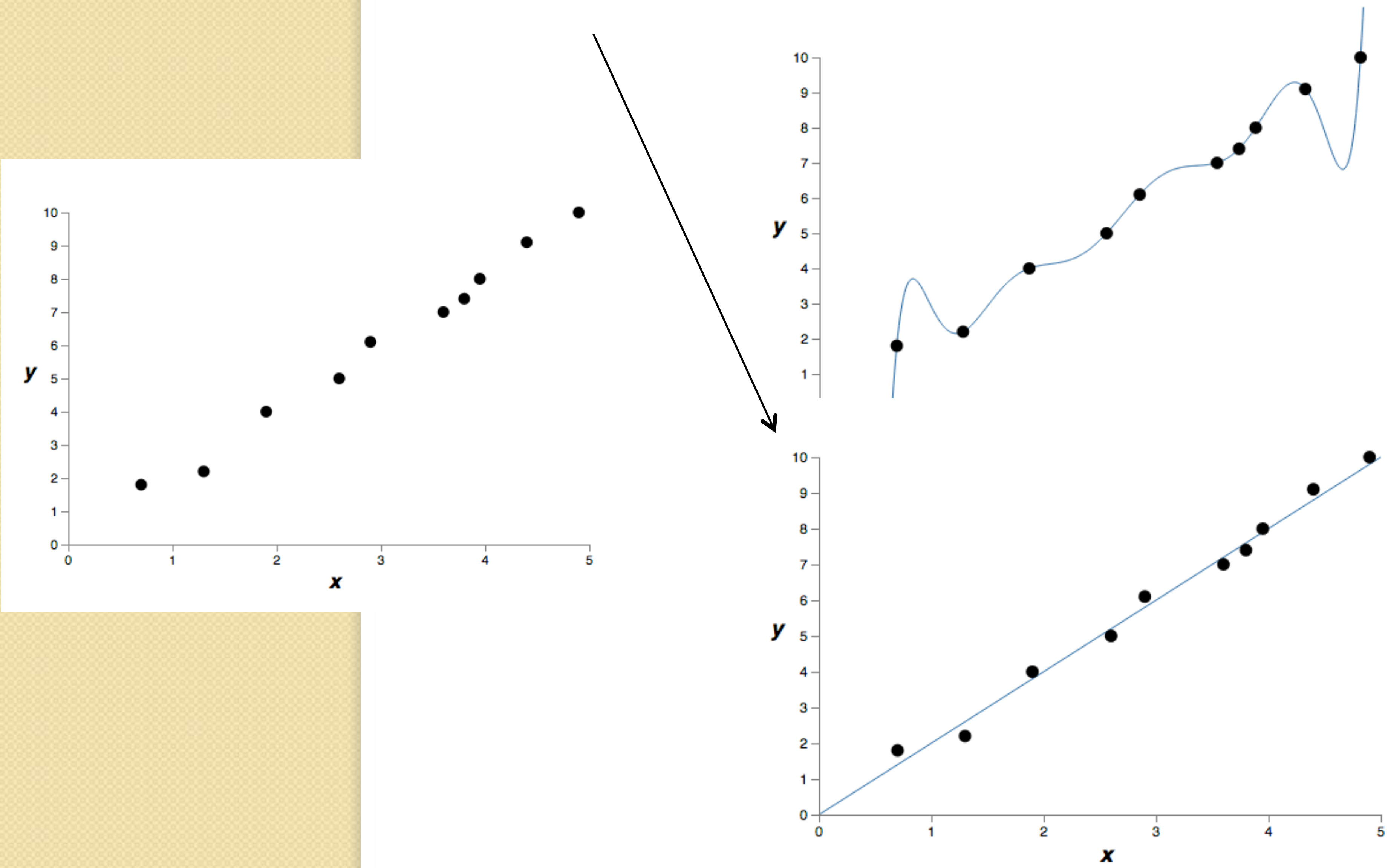
Compromisso entre redução do erro e pesos pequenos

Efeito do Parâmetro λ



The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization. You can play with these examples in this ConvNetsJS demo.

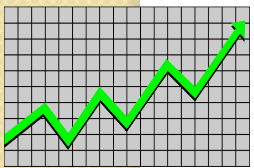
Efeitos da Regularização



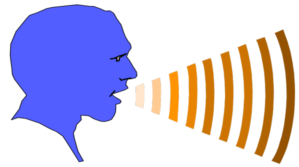
Demo…

http://playground.tensorflow.org/

Aplicações do MLP



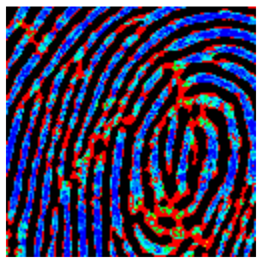
Análise de mercado

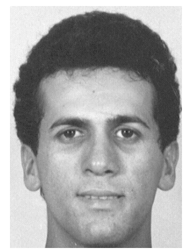


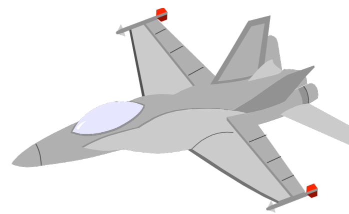
Proc. voz

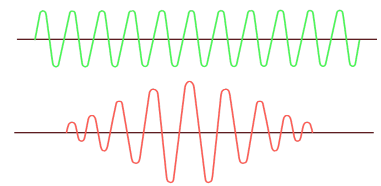
Data mining

Análise de crédito

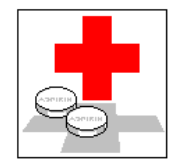






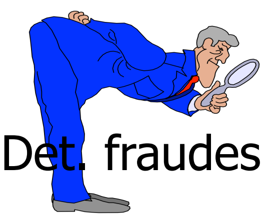


Proc. sinais

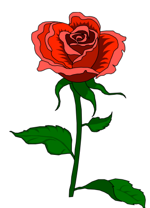


Diagnose médica

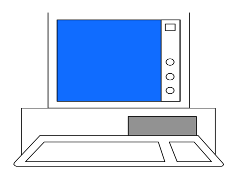
Previsão séries



Det. fraudes



Rec. odores



Interfaces

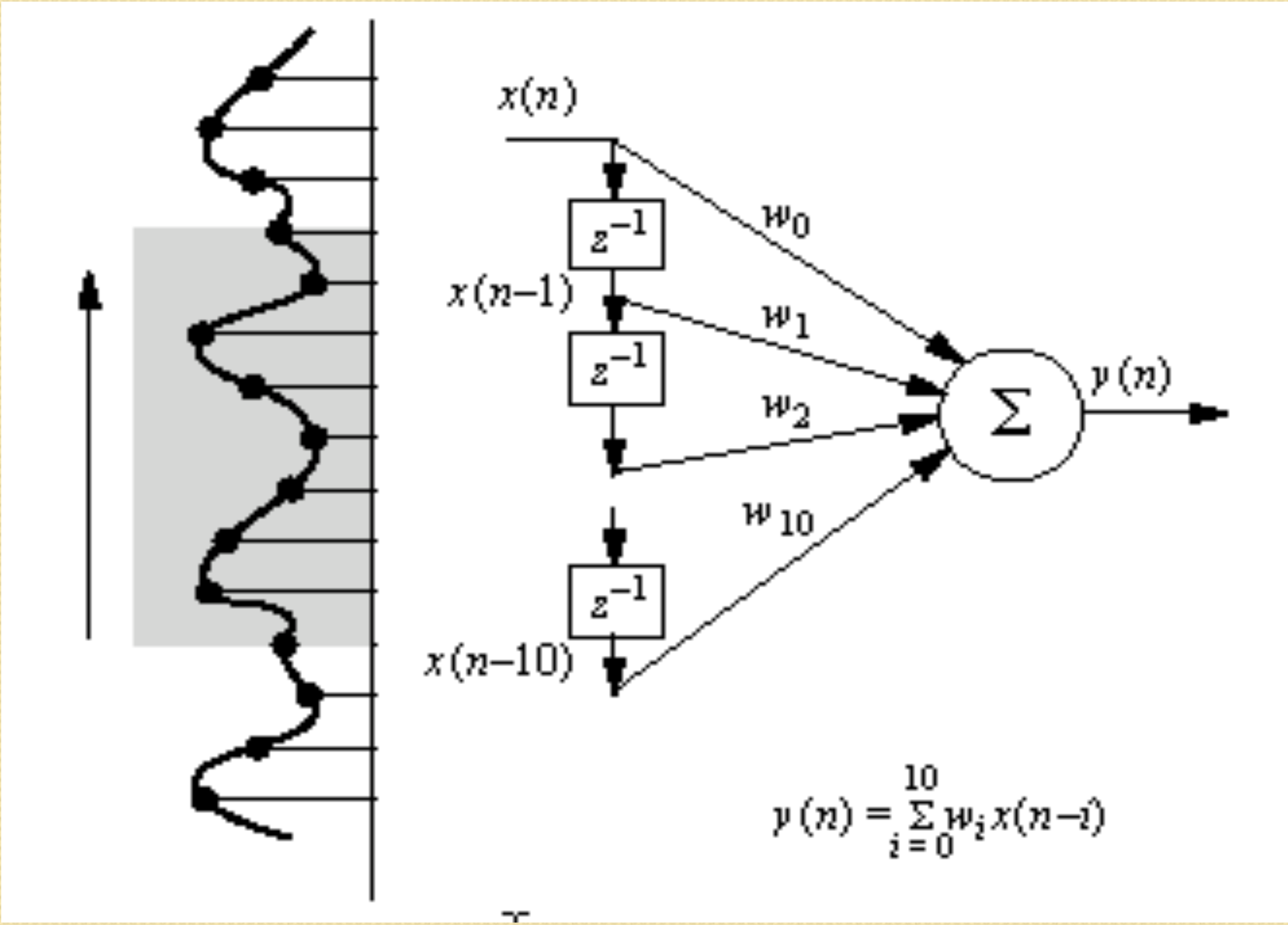
Exemplo: Previsão (Forecasting)

- Dado um conjunto de n valores de uma variável ($y(t_1)$, $y(t_2)$, ..., $y(t_n)$) em uma sequência de tempo t_1, t_2, ...,t_n,
- Prever o valor $y(t_{n+1})$ num futuro t_{n+1}

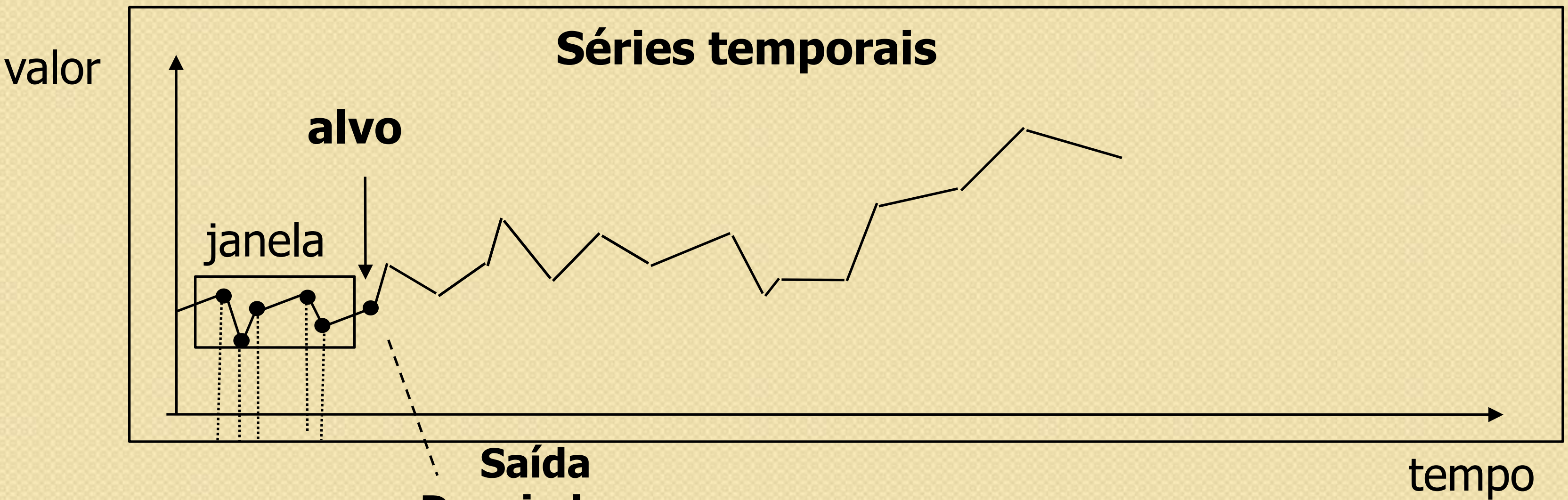
Previsão – Questões Relevantes

- Definição da janela de entrada
- Definição do horizonte de previsão
- Definição de outras variáveis explicativas

Previsão com uma Rede MLP



Previsão de Séries Temporais



Saída Desejada = valor da série k passos à frente

Entradas da rede = n valores passados

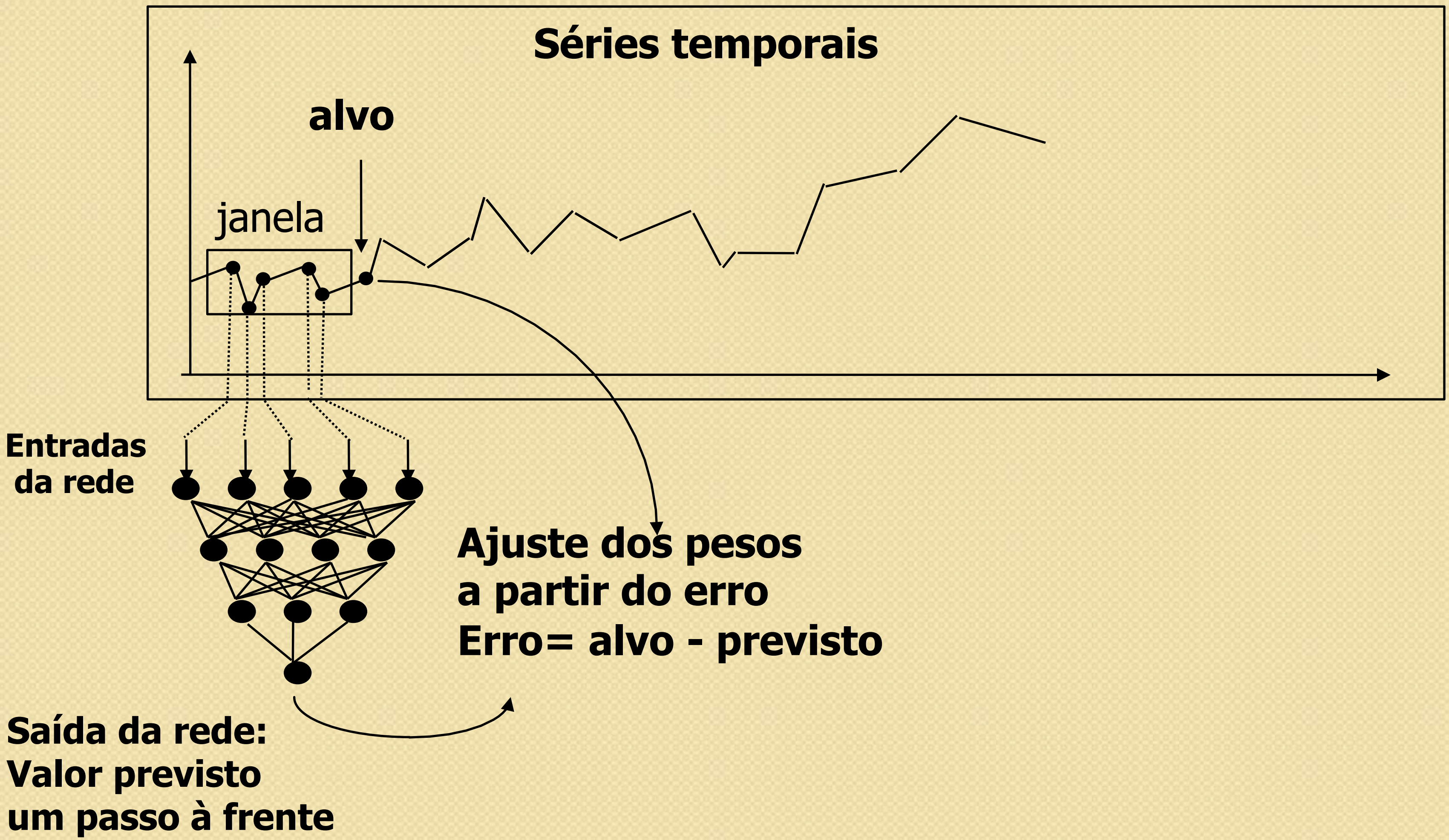
Definição da janela de entrada

Definição da janela de saída

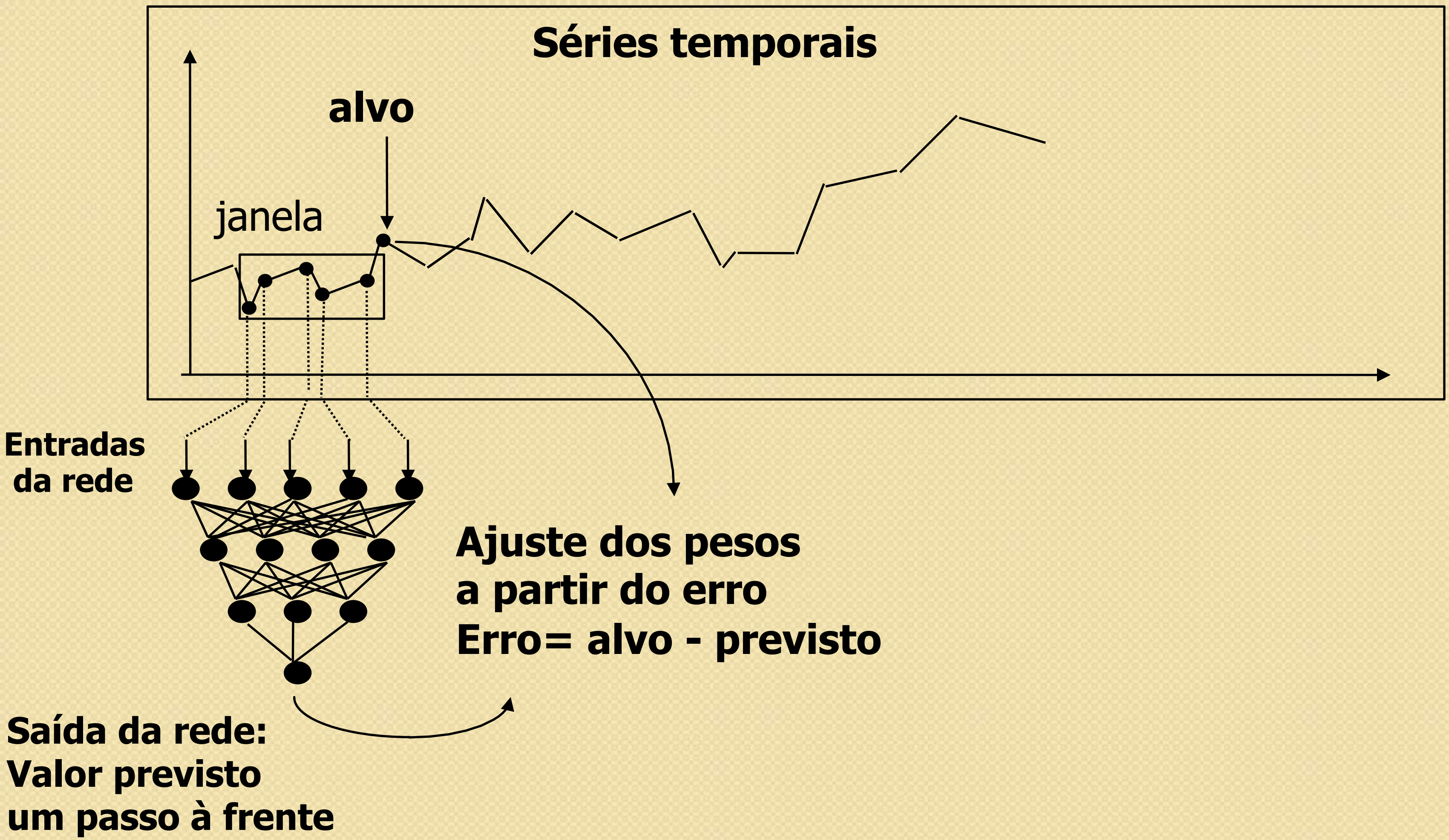
Ex: 5 valores passados

Ex: valor um passo à frente

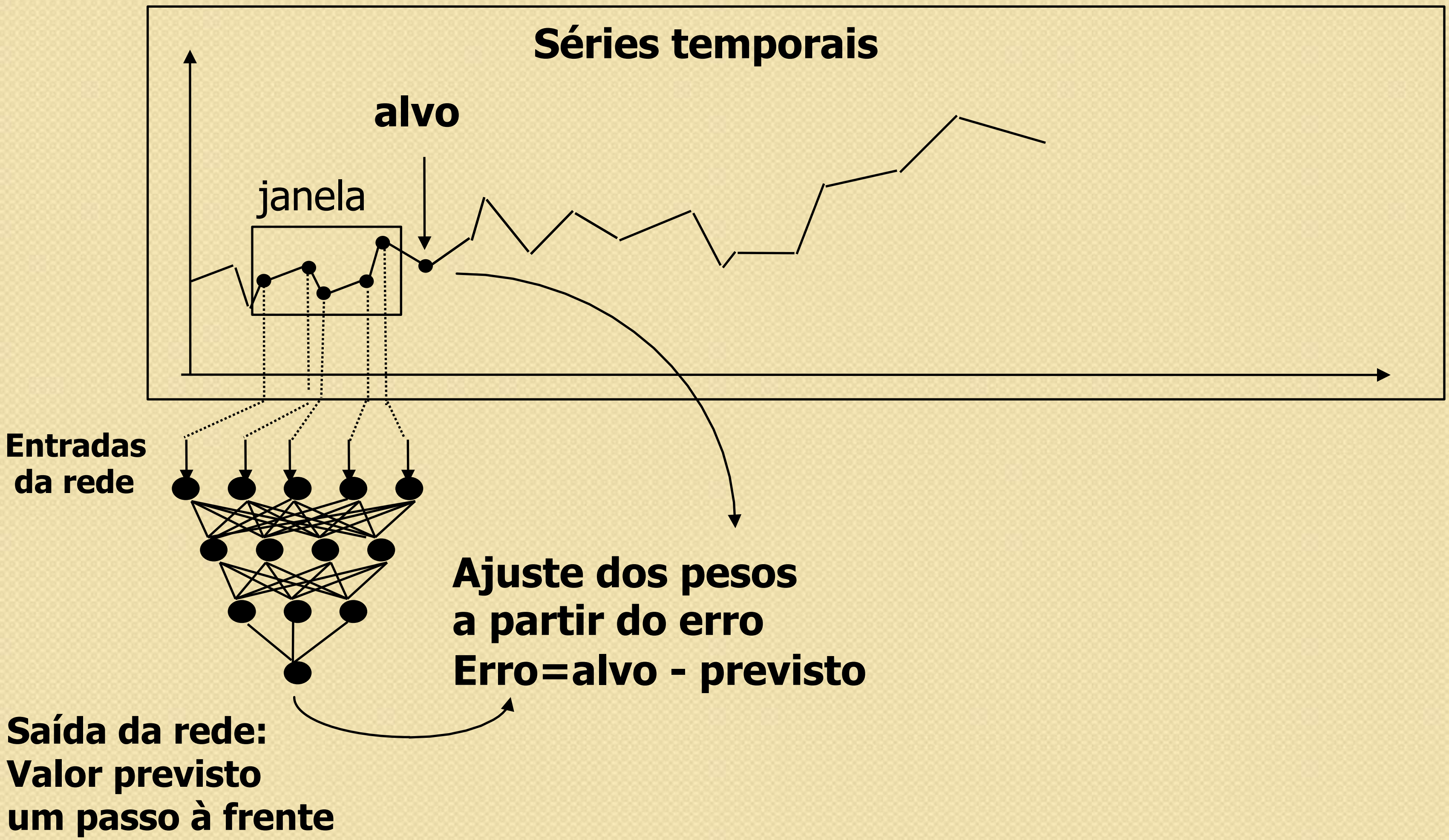
Exemplo: previsão utilizando apenas a série histórica como entrada



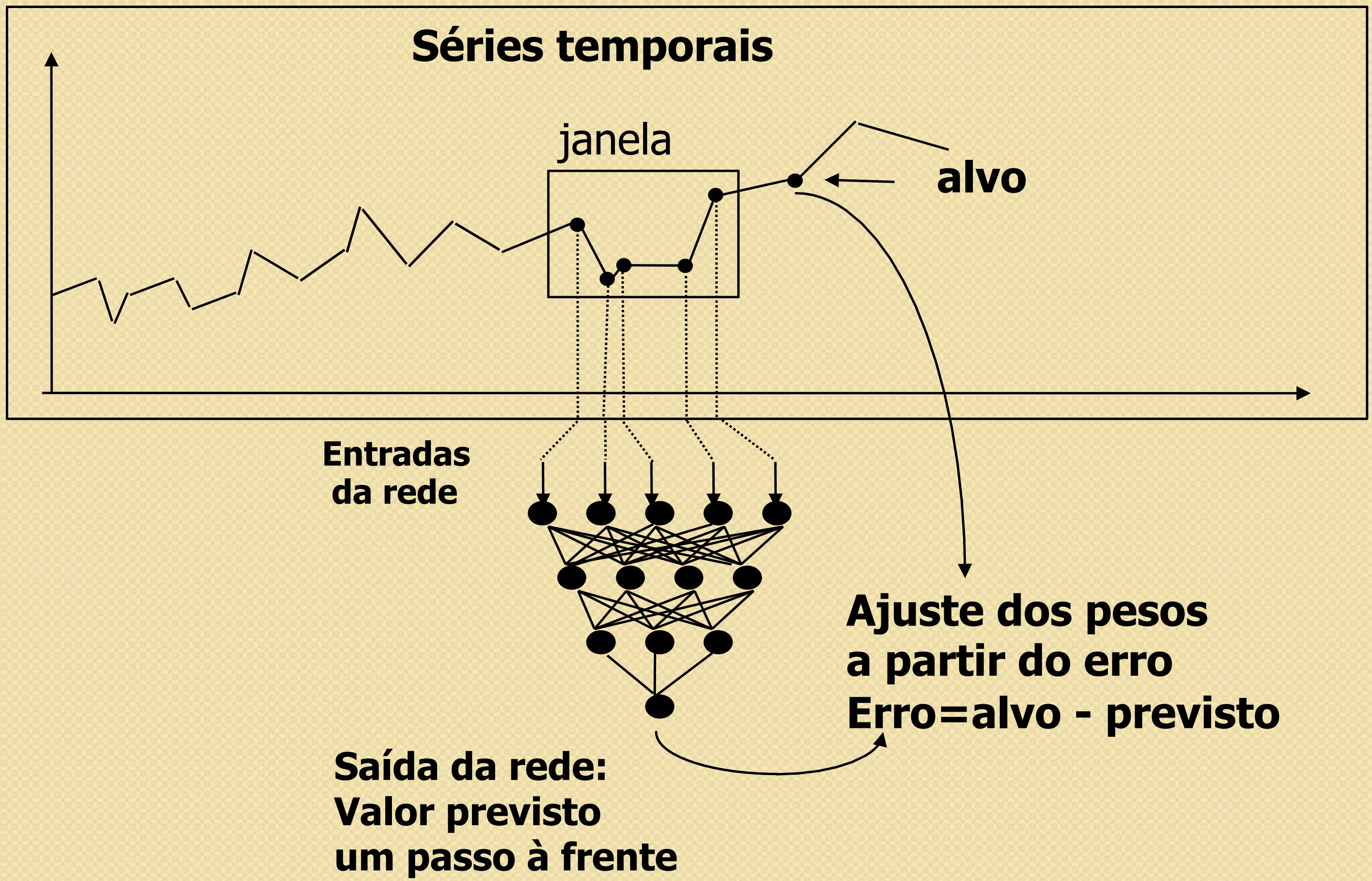
Exemplo: previsão utilizando apenas a série histórica como entrada



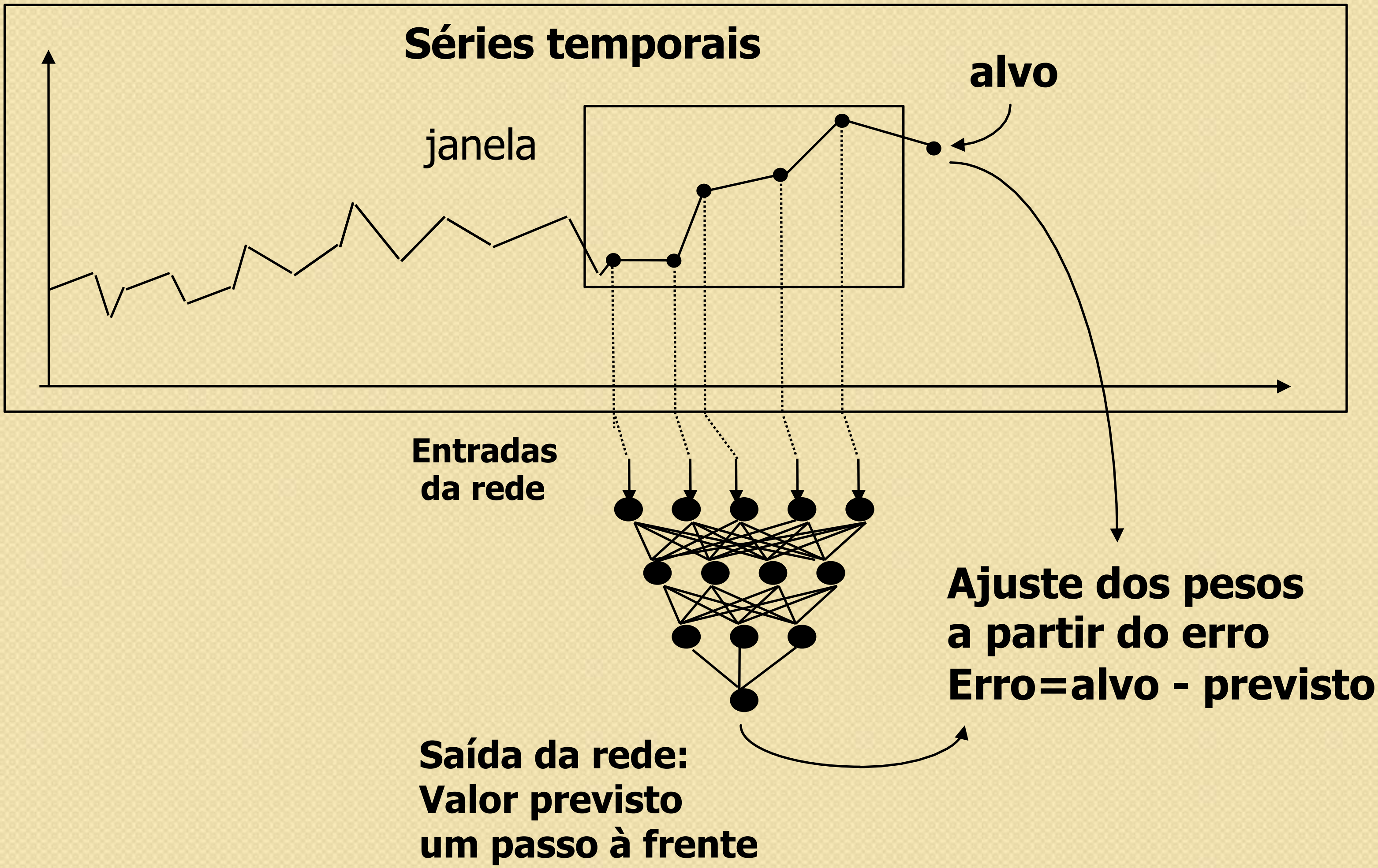
Exemplo: previsão utilizando apenas a série histórica como entrada



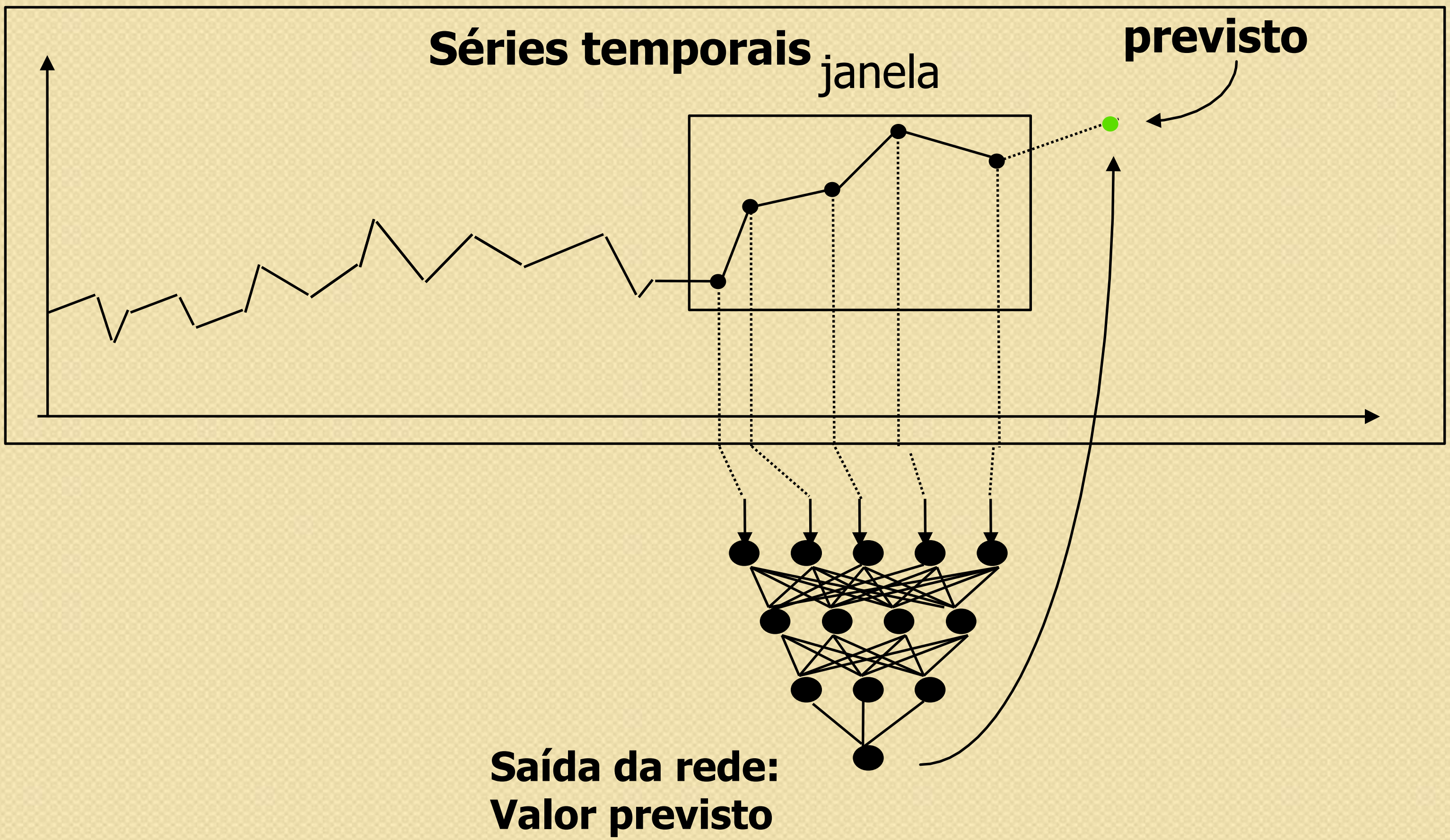
Exemplo: previsão utilizando apenas a série histórica como entrada



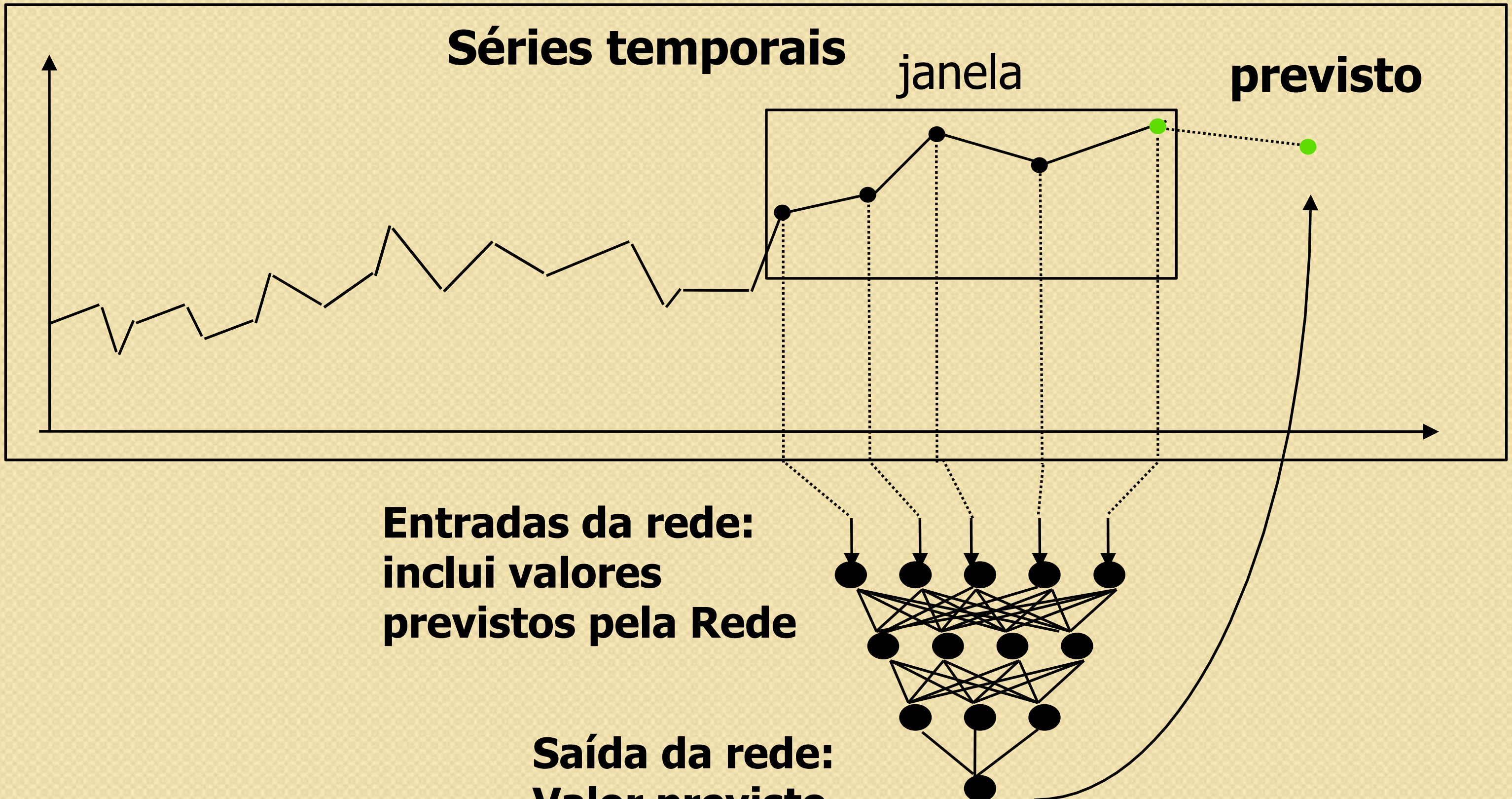
Exemplo: previsão utilizando apenas a série histórica como entrada



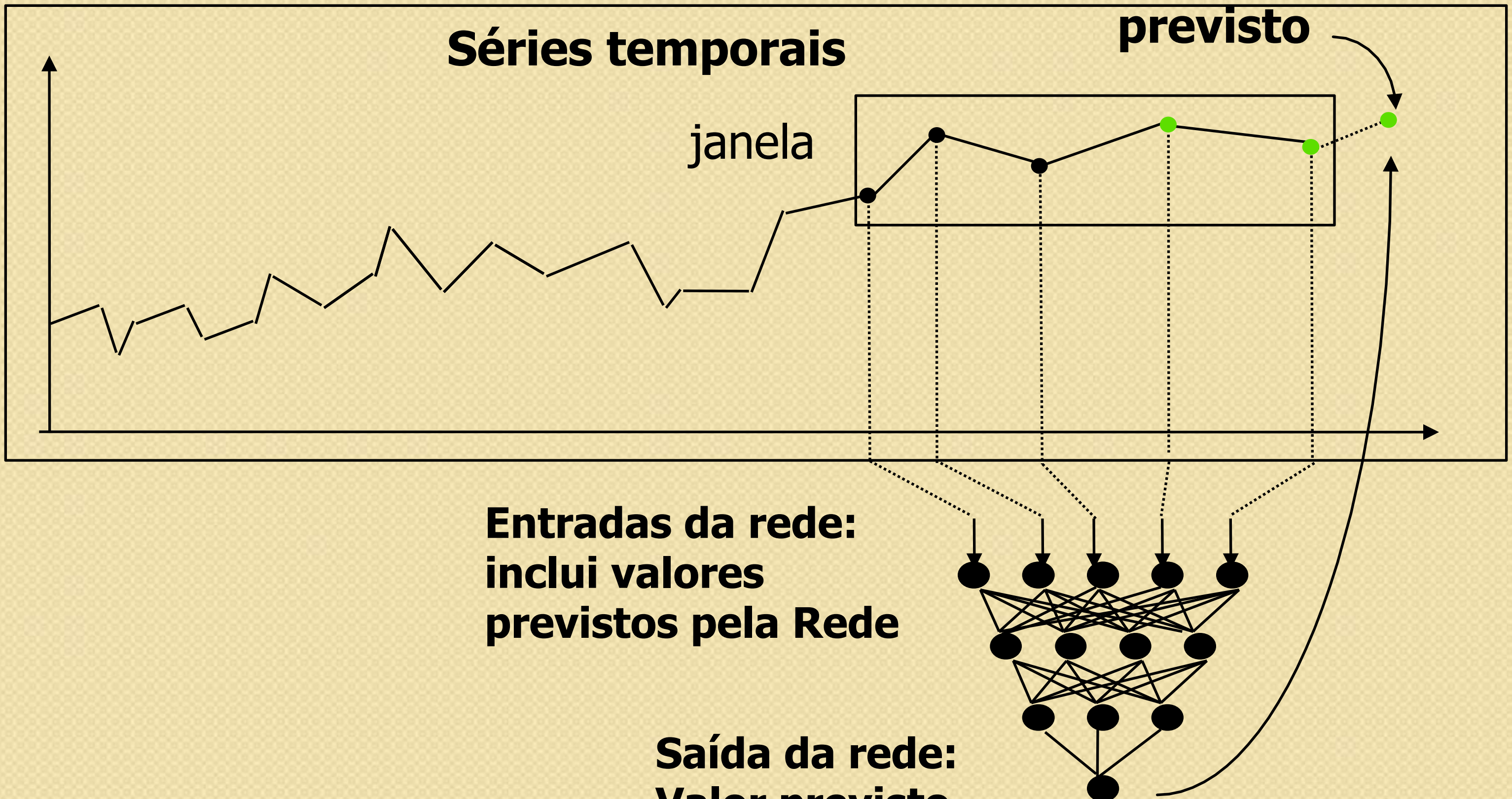
Exemplo: previsão utilizando apenas a série histórica como entrada



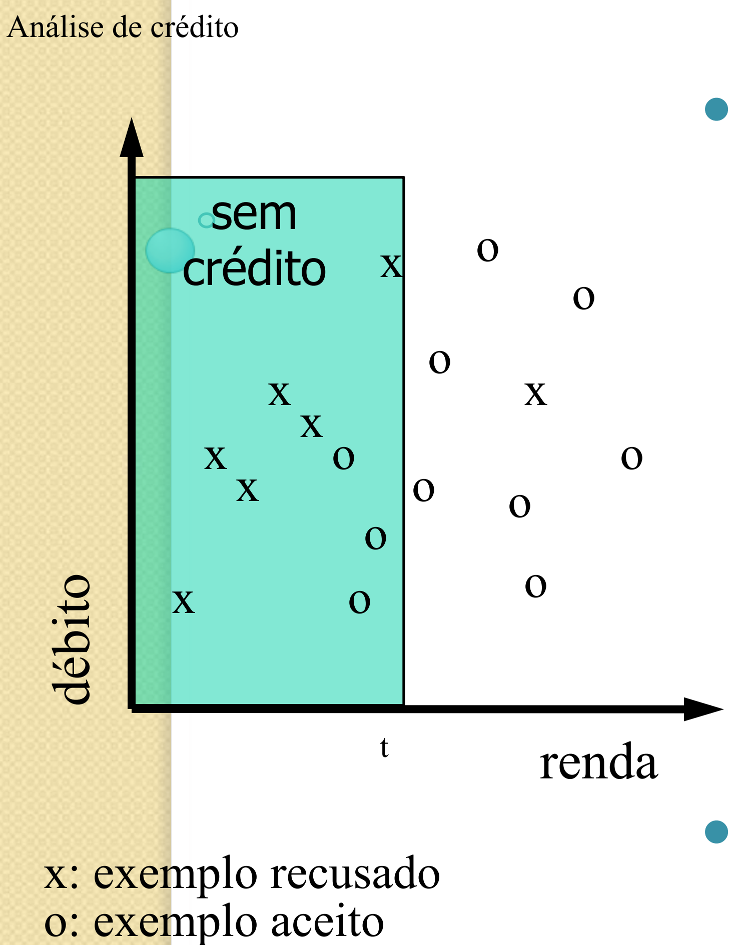
Exemplo: previsão utilizando apenas a série histórica como entrada



Exemplo: previsão utilizando apenas a série histórica como entrada

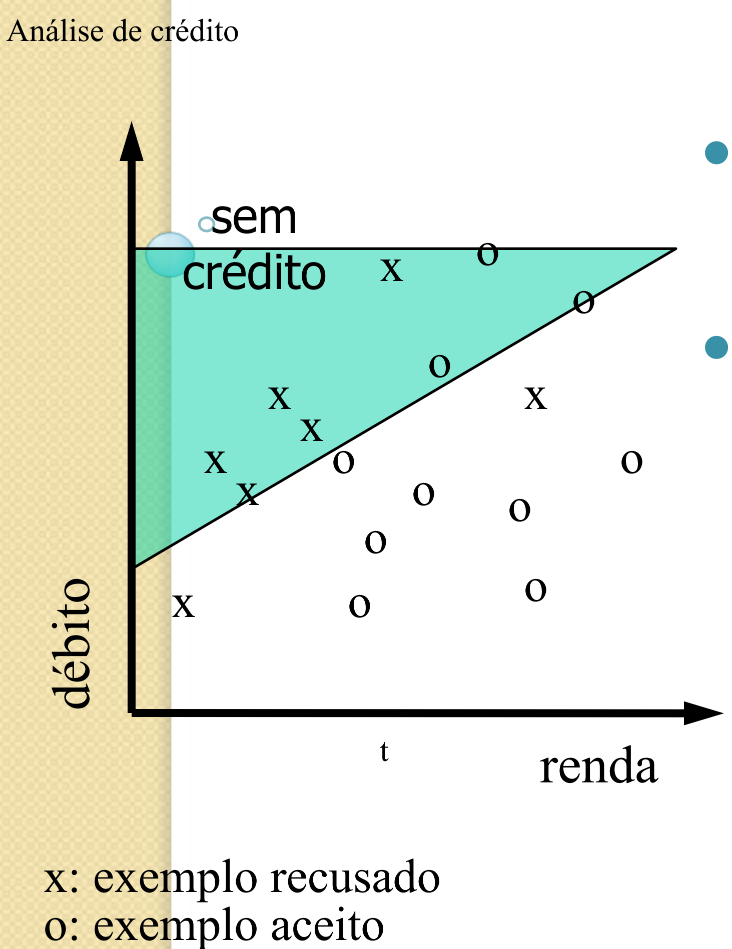


Complexidade Funcional (I)



- Um hiperplano paralelo de separação: pode ser interpretado diretamente como uma regra:
  - se a renda é menor que t, então o crédito não deve ser liberado
- Exemplo:
  - árvores de decisão;
  - indução de regras

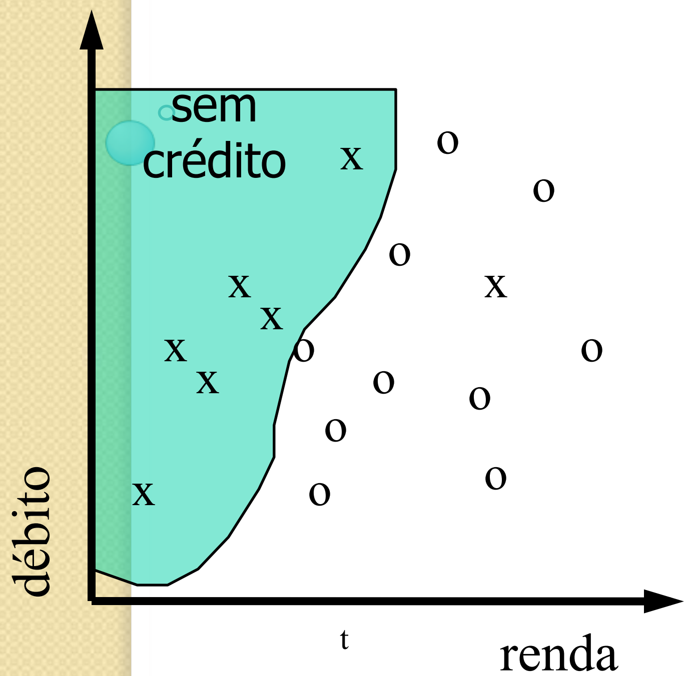
Complexidade Funcional (II)



- Hiperplano oblíquo: melhor separação:
- Exemplos:
  - regressão linear;
  - perceptron;

Complexidade Funcional (III)

Análise de crédito

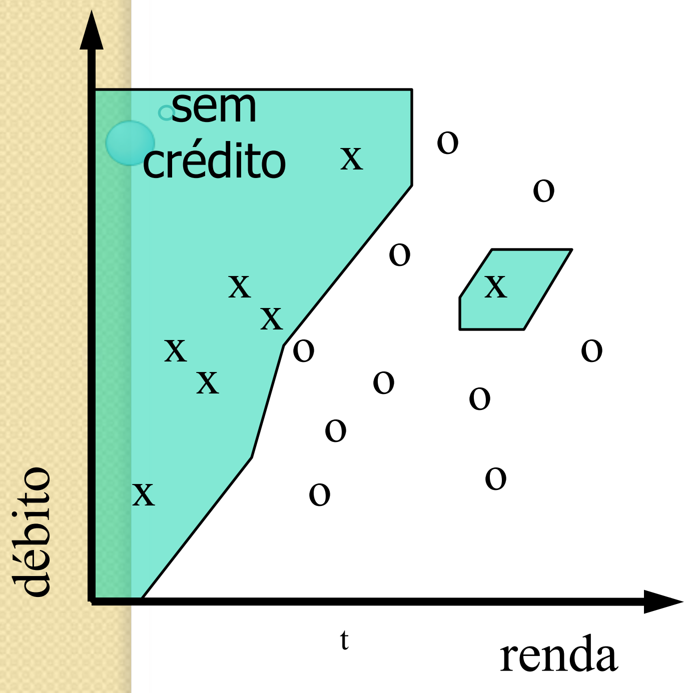


x: exemplo recusado
o: exemplo aceito

- Superfície não linear: melhor poder de classificação, pior interpretação;
- Exemplos:
  - perceptrons multicamadas;
  - regressão não-linear;

Complexidade Funcional (IV)

Análise de crédito

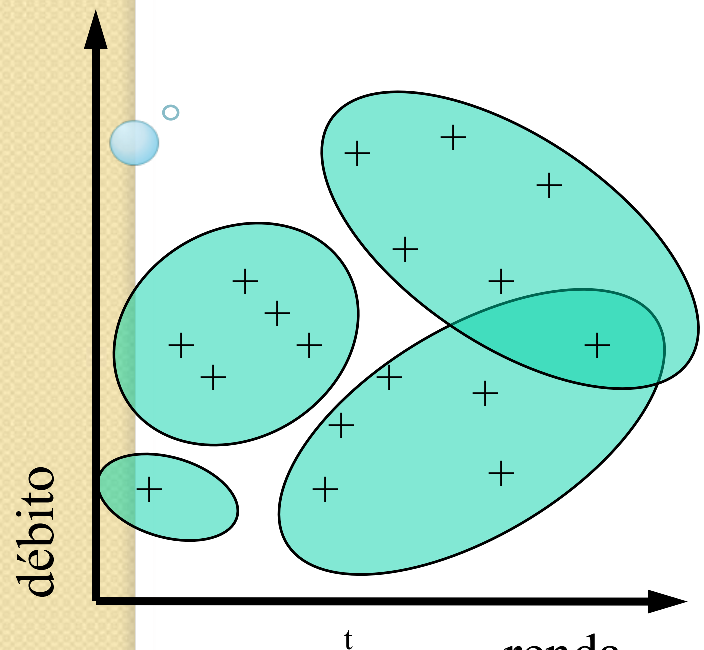


x: exemplo recusado
o: exemplo aceito

- Métodos baseado em exemplos;
- Exemplos:
  - k-vizinhos mais próximos;
  - raciocínio baseado em casos;
  - perceptrons multicamadas

Complexidade Funcional (V)

Análise de crédito



+: exemplo

- Agrupamento
- Exemplo:
  - vector quantization;
  - ART (Adaptive Ressonance Theory)