Inatel
Instituto Nacional de Telecomunicações

www.inatel.br

Redes Neurais Artificiais

Edielson Prevato Frigieri

edielson@inatel.br
Av. João de Camargo, 510
Santa Rita do Sapucai - MG
Tel: (35) 3471-9913

Julho 2014

Inatel
Instituto Nacional de Telecomunicações

www.inatel.br

Redes Neurais Artificiais

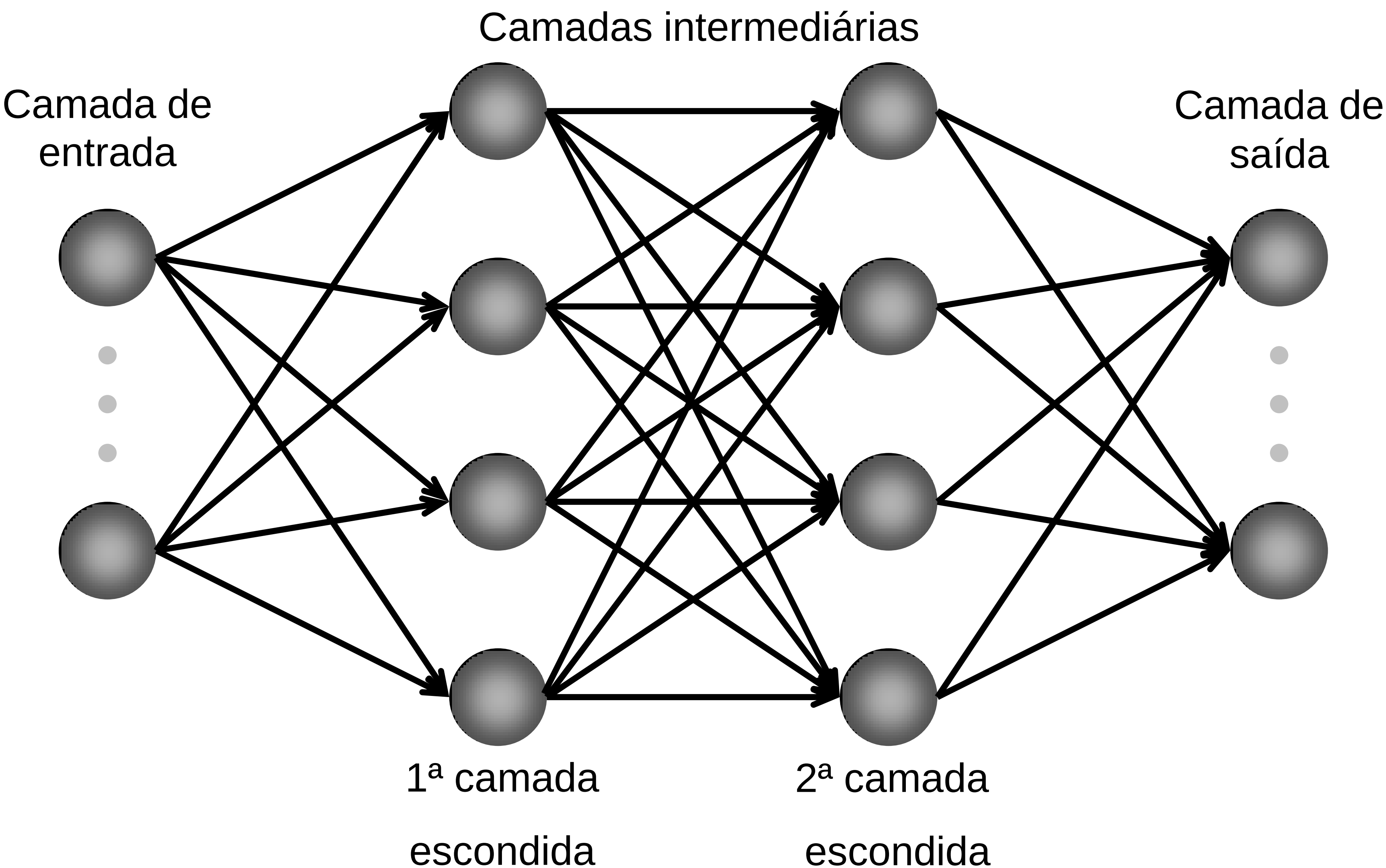
Redes *Perceptron* Multicamadas (*MLP – Multilayer Perceptron*)

Julho 2014

Perceptron multicamadas

- Caracteriza-se pela presença de pelo menos uma camada intermediária (escondida);
- Podem ser aplicadas em diversos tipos de problemas:
  - ✓ Aproximação universal de funções;
  - ✓ Reconhecimento de padrões;
  - ✓ Identificação e controle de processos;
  - ✓ Previsão de séries temporais;
  - ✓ Otimização de sistemas.
- Possui arquitetura *feedforward* de camadas múltiplas;
- Tornou-se famosa após a apresentação do algoritmo de aprendizagem *backpropagation*, apresentado por Rumelhart (1986).

Perceptron multicamadas, cont.



Princípio de funcionamento

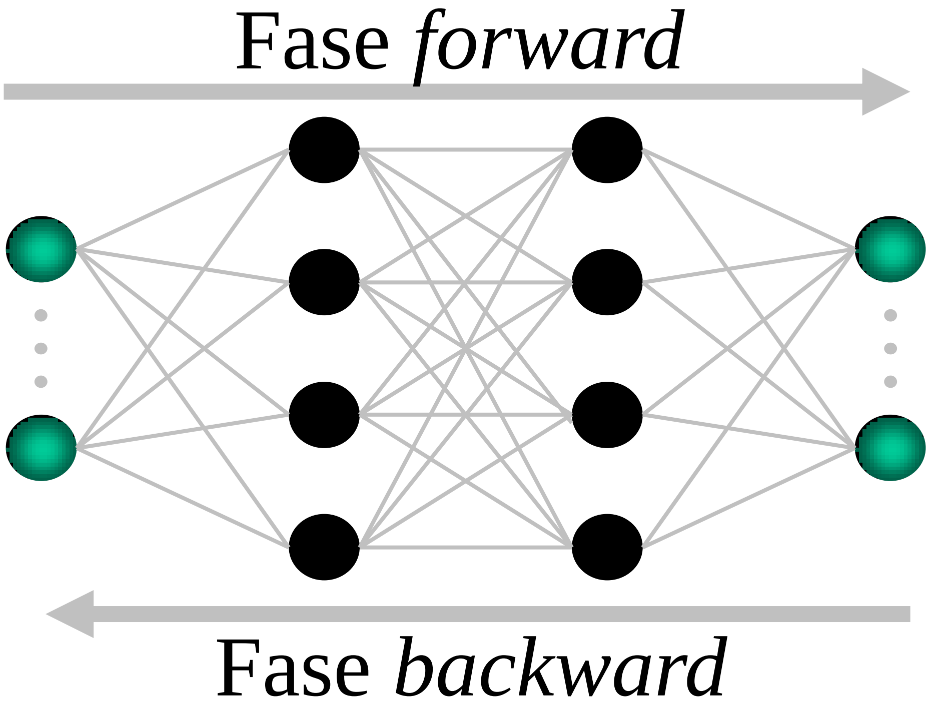
- Cada uma das entradas da rede será propagada em direção à camada neural de saída;
- As saídas dos neurônios da primeira camada escondida serão as entradas dos neurônios da segunda camada escondida;
- As saídas da última camada escondida serão as respectivas entradas da camada de saída;
- A propagação dos sinais de entrada ocorre sempre em um único sentido: entrada para saída;
- Além da presença de camadas escondidas, a rede MLP pode apresentar uma camada de saída composta por vários neurônios, sendo um neurônio para cada processo a ser mapeado;

Princípio de funcionamento, cont.

- O conhecimento relacionado ao comportamento entrada/saída do sistema será distribuído por todos os neurônios da rede MLP;
- A definição da configuração topológica de uma rede MLP de depende de suma série de fatores:
  - Disposição espacial das amostras de treinamento;
  - Valoreis iniciais atribuídos ao parâmetros de treinamento;
  - Valoreis iniciais atribuídos ás matrizes de pesos sinápticos.
- O ajuste dos pesos e do limiar de cada um dos neurônios é efetuado utilizando-se um processo de treinamento supervisionado;
- O algoritmos de aprendizado utilizado é denominado *backpropagation*;

Processo de treinamento

- A regra de aprendizagem baseada na correção do erro pelo método do gradiente;
- O treinamento é feito em duas fases: *forward* (cálculo do erro) e *backward* (correção dos pesos sinápticos);

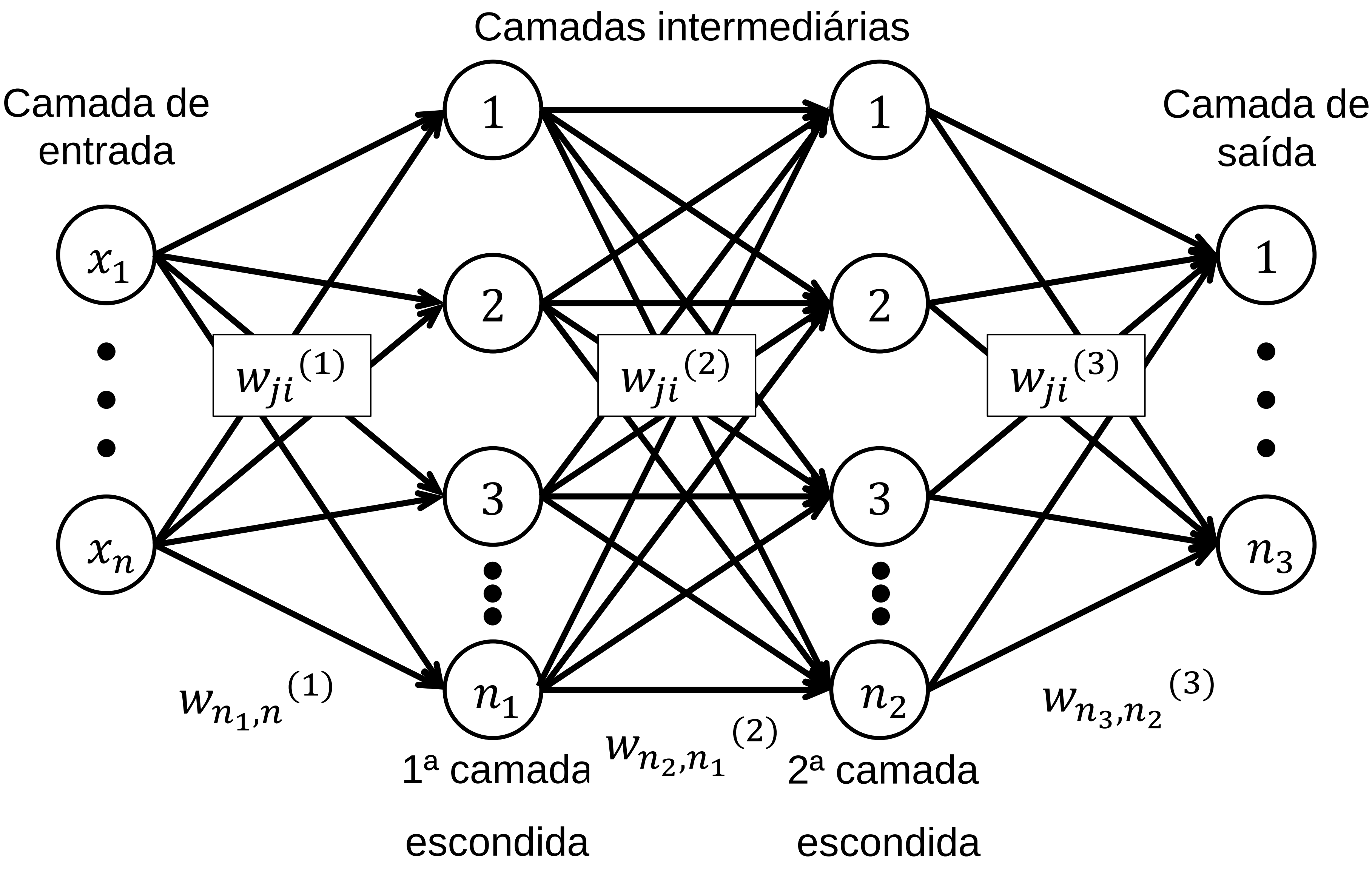


Algoritmo *Backpropagation*

- Durante o treinamento com o algoritmo *backpropagation*, a rede opera em uma sequência de dois passos:

  - Primeiro, um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída.
  - Segundo passo, a saída obtida é comparada à saída desejada para esse padrão particular. Se esta não estiver correta, o erro é calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retro propagado.

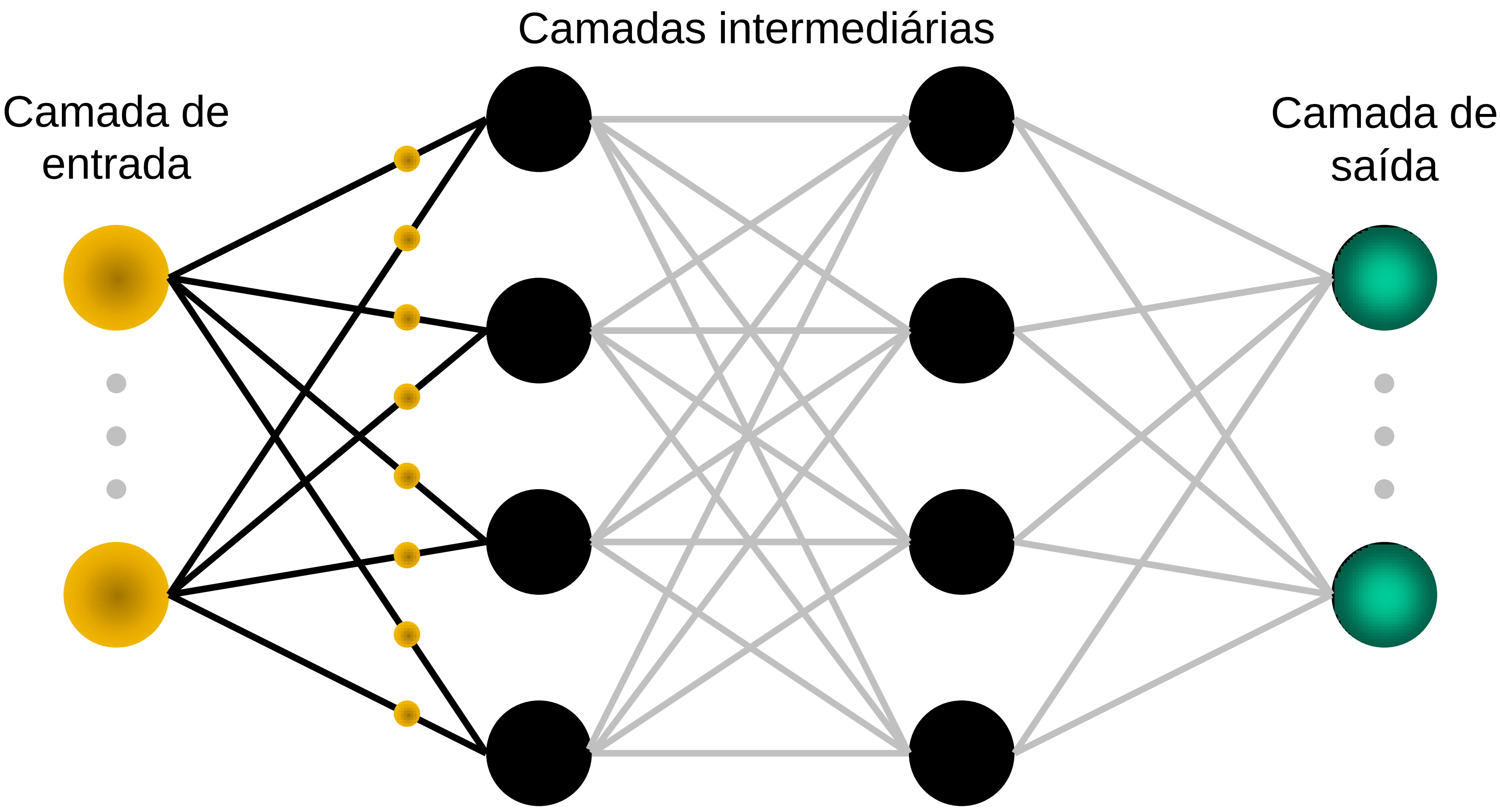
Algoritmo *Backpropagation*, cont.



Algoritmo *Backpropagation*, cont.

- Fase *forward*

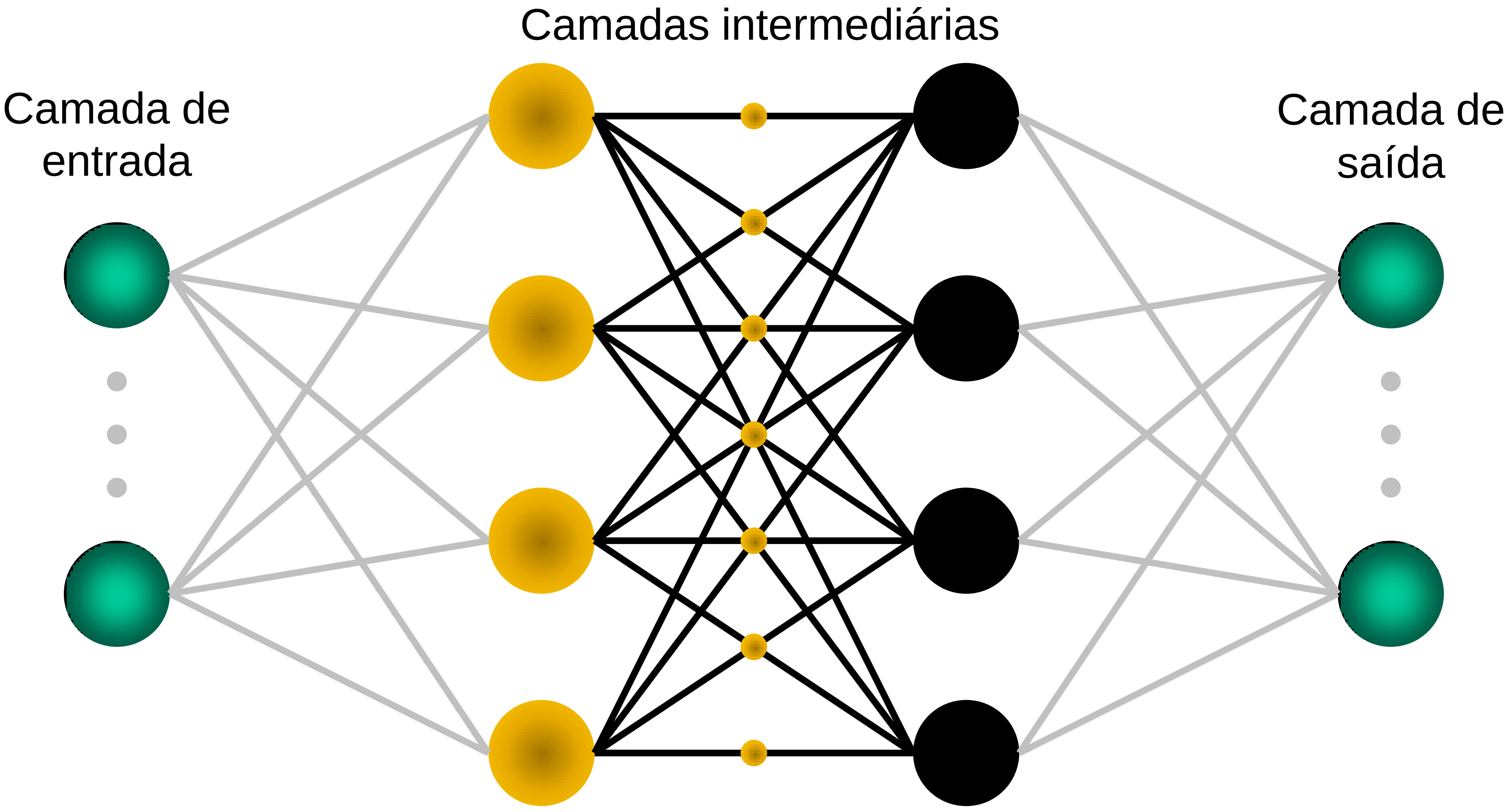
Entrada é apresentada à primeira camada da rede e é propagado em direção às saídas.



Algoritmo *Backpropagation*, cont.

- Fase *forward*

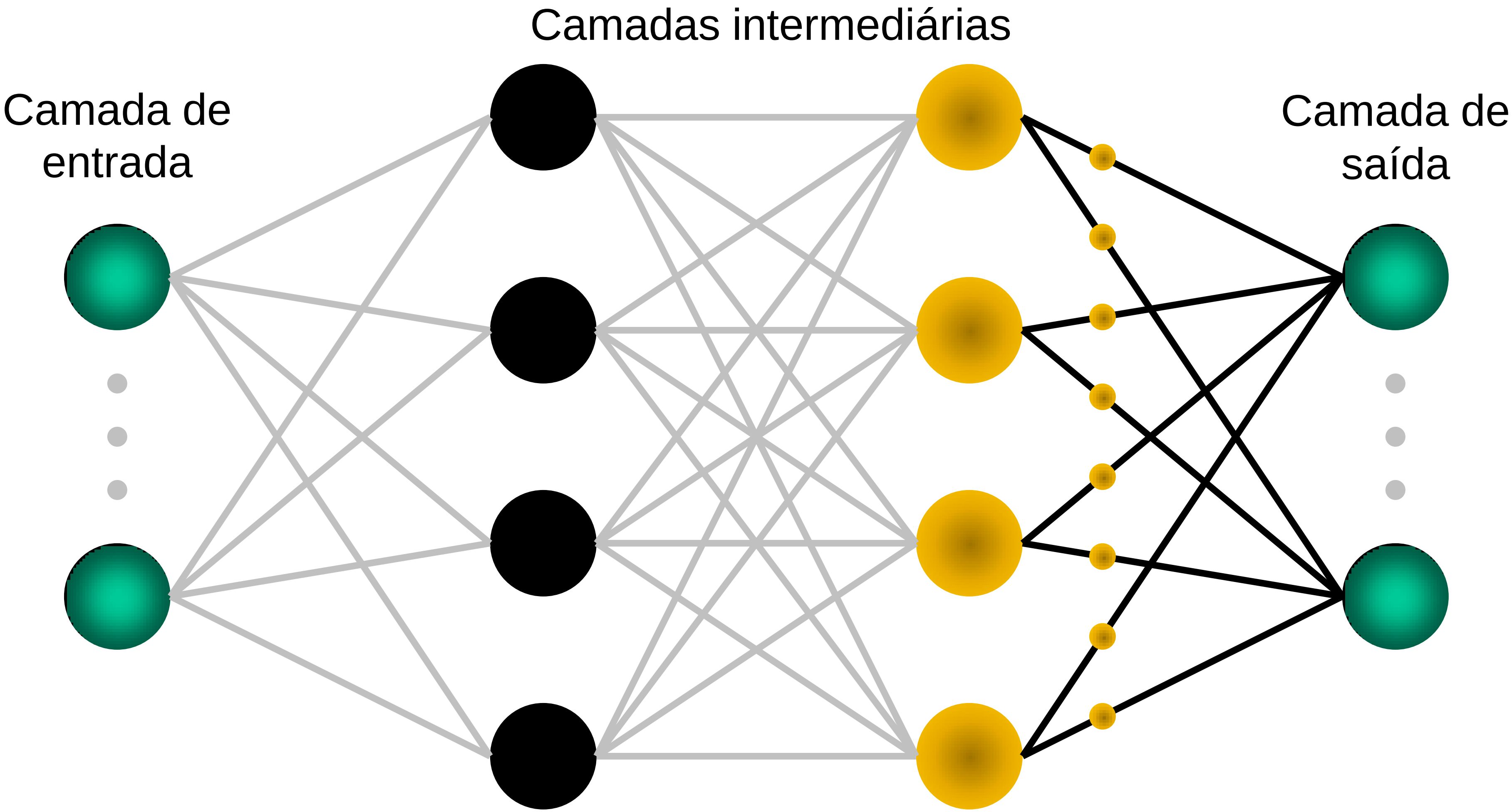
Os neurônios da camada i calculam seus sinais de saída e propagam à camada $i + 1$



Algoritmo *Backpropagation*, cont.

- Fase *forward*

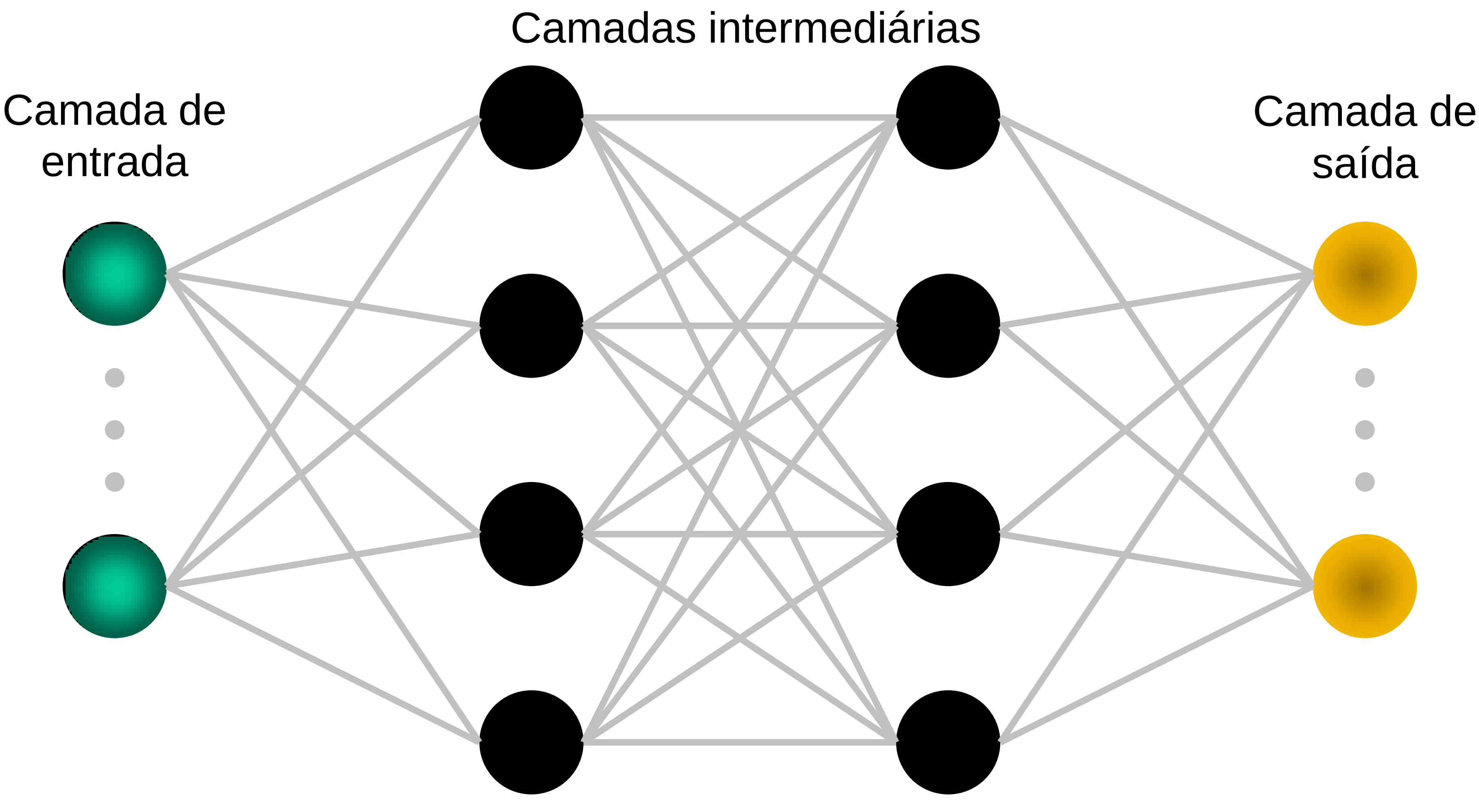
A última camada oculta calcula seus sinais de saída e os envia à camada de saída



Algoritmo *Backpropagation*, cont.

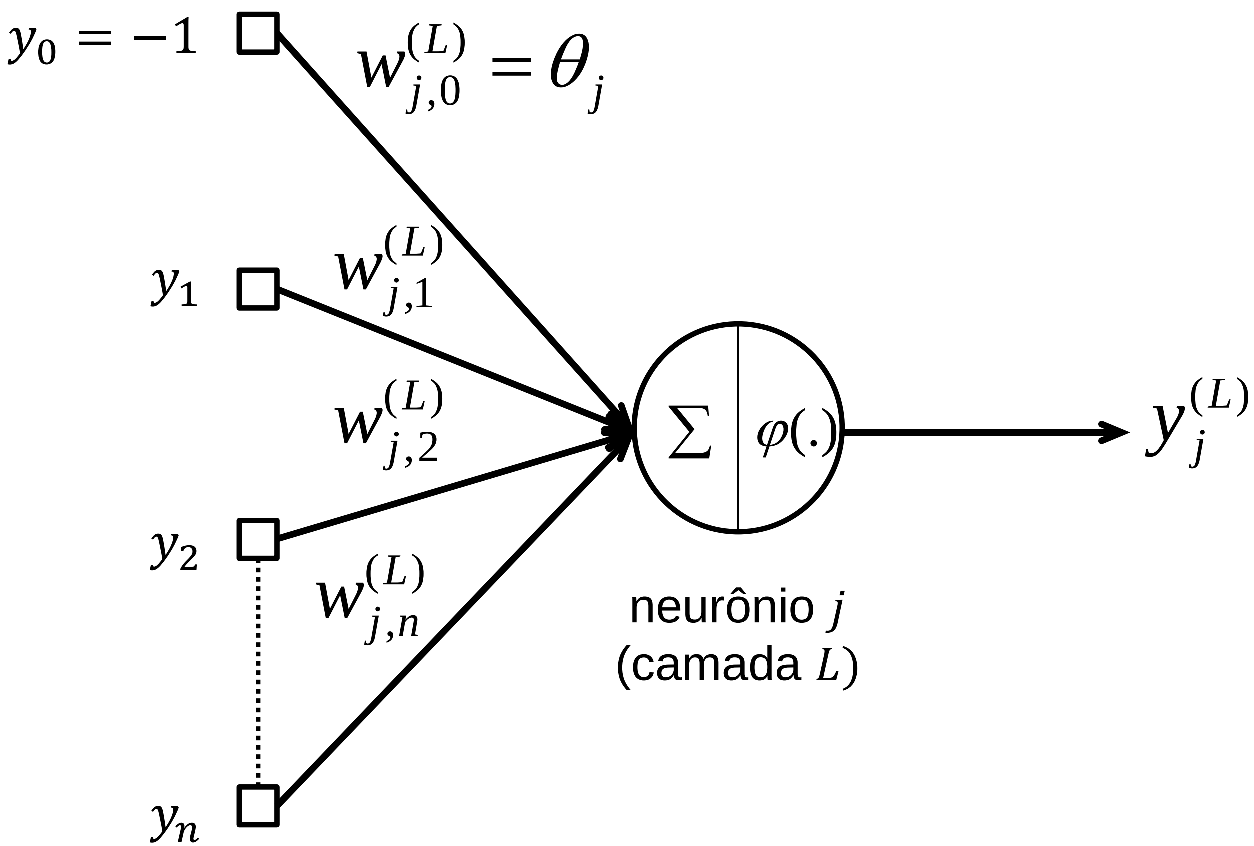
➢ Fase *forward*

A camada de saída calcula os valores de saída da rede.



Algoritmo *Backpropagation*, cont.

- Neurônio de saída



Algoritmo *Backpropagation*, cont.

- Podemos calcular o erro associado a saída y de um neurônio j, em um instante k, através da equação:

$$e_j(k) = (d_j(k) - y_j(k))$$

- Logo, para calcular o desvio entre as respostas produzidas e os valores desejados, pode-se utilizar a seguinte equação de erro quadrático para cada saída:

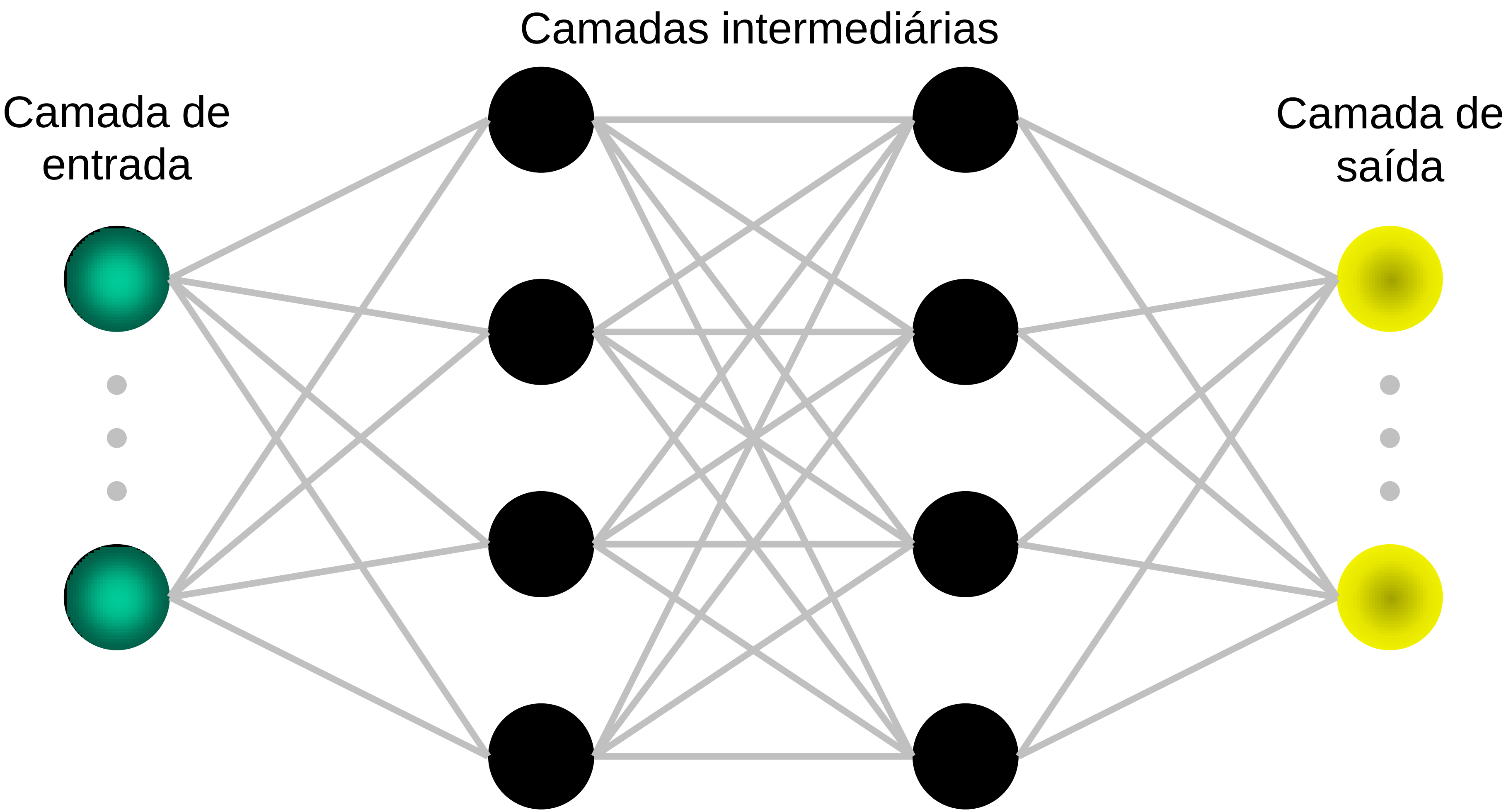
$$E(k) = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2(k) = \frac{1}{2}\sum_{j=1}^{n_L}(d_j(k) - y_j^{(L)}(k))^2$$

- Considerando um conjunto de p amostras, o erro quadrático médio fica:

$$E_{qm} = \frac{1}{\mathrm{p}}\sum_{k=1}^{p} E(k)$$

Algoritmo *Backpropagation*, cont.

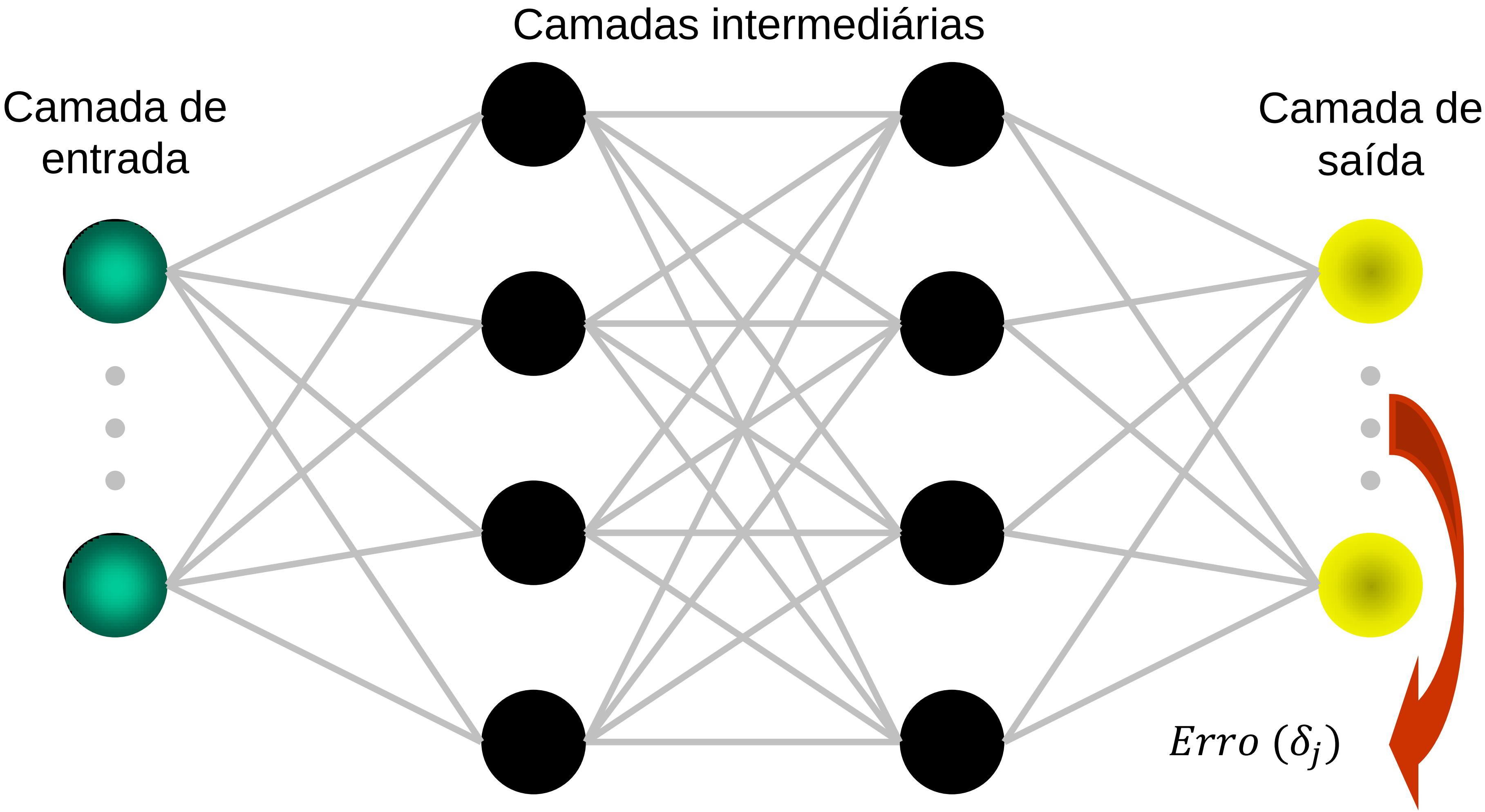
- Fase *backward*



Algoritmo *Backpropagation*, cont.

- **Fase *backward***

A camada de saída calcula o erro da rede δ_j.



Algoritmo *Backpropagation*, cont.

$$\Delta E^{(L)} = \frac{\partial E(k)}{\partial \omega_{ji}^{(L)}(k)} = \frac{\partial E(k)}{\partial y_j^{(L)}(k)} \frac{\partial y_j^{(L)}(k)}{\partial v_j^{(L)}(k)} \frac{\partial v_j^{(L)}(k)}{\partial \omega_{ji}^{(L)}(k)}$$

$$E(k) = \frac{1}{2}\sum_{j=1}^{n_L}(d_j(k) - y_j^{(L)}(k))^2 \Rightarrow \frac{\partial E(k)}{\partial y_j^{(L)}(k)} = -(d_j(k) - y_j^{(L)}(k))$$

$$y_j^{(L)}(k) = \varphi\left(v_j^{(L)}(k)\right) \Rightarrow \frac{\partial y_j^{(L)}(k)}{\partial v_j^{(L)}(k)} = \varphi'\left(v_j^{(L)}(k)\right)\frac{\partial v_j^{(L)}(k)}{\partial v_j^{(L)}(k)} = \varphi'\left(v_j^{(L)}(k)\right)$$

$$v_j^{(L)}(k) = \sum_{i=0}^{n_{L-1}} \omega_{ji}^{(L)}(k) y_i^{(L-1)}(k) \Rightarrow \frac{\partial v_j^{(L)}(k)}{\partial w_{ji}^{(L)}(k)} = y_i^{(L-1)}(k)$$

Algoritmo *Backpropagation*, cont.

- Desta forma obtermos:

$$\frac{\partial E(k)}{\partial \omega_{ji}^{(L)}(k)} = -e_{j}^{(L)}(k)\varphi'(v_{j}^{(L)}(k))y_{i}^{(L-1)}(k)$$

- Partindo da regra delta:

$$\Delta\omega_{ji}^{(L)}(k) = \eta e_{j}^{(L)}(k) y_{i}^{(L-1)}(k) = -\eta \frac{\partial E(k)}{\partial \omega_{ji}^{(L)}(k)}$$

- Chegamos:

$$\Delta\omega_{ji}^{(L)}(k) = \eta \delta_{j}^{(L)}(k) y_{i}^{(L-1)}(k)$$

Algoritmo *Backpropagation*, cont.

- Onde o gradiente local $\delta_j(n)$ é dado por:

$$\delta_j^{(L)}(k) = -\frac{\partial E(k)}{\partial \omega_{ji}^{(L)}(k)} = e_j^{(L)}(k)\varphi'(v_j^{(L)}(k))$$

- Basta ajustar os pesos sinápticos da ultima camada:

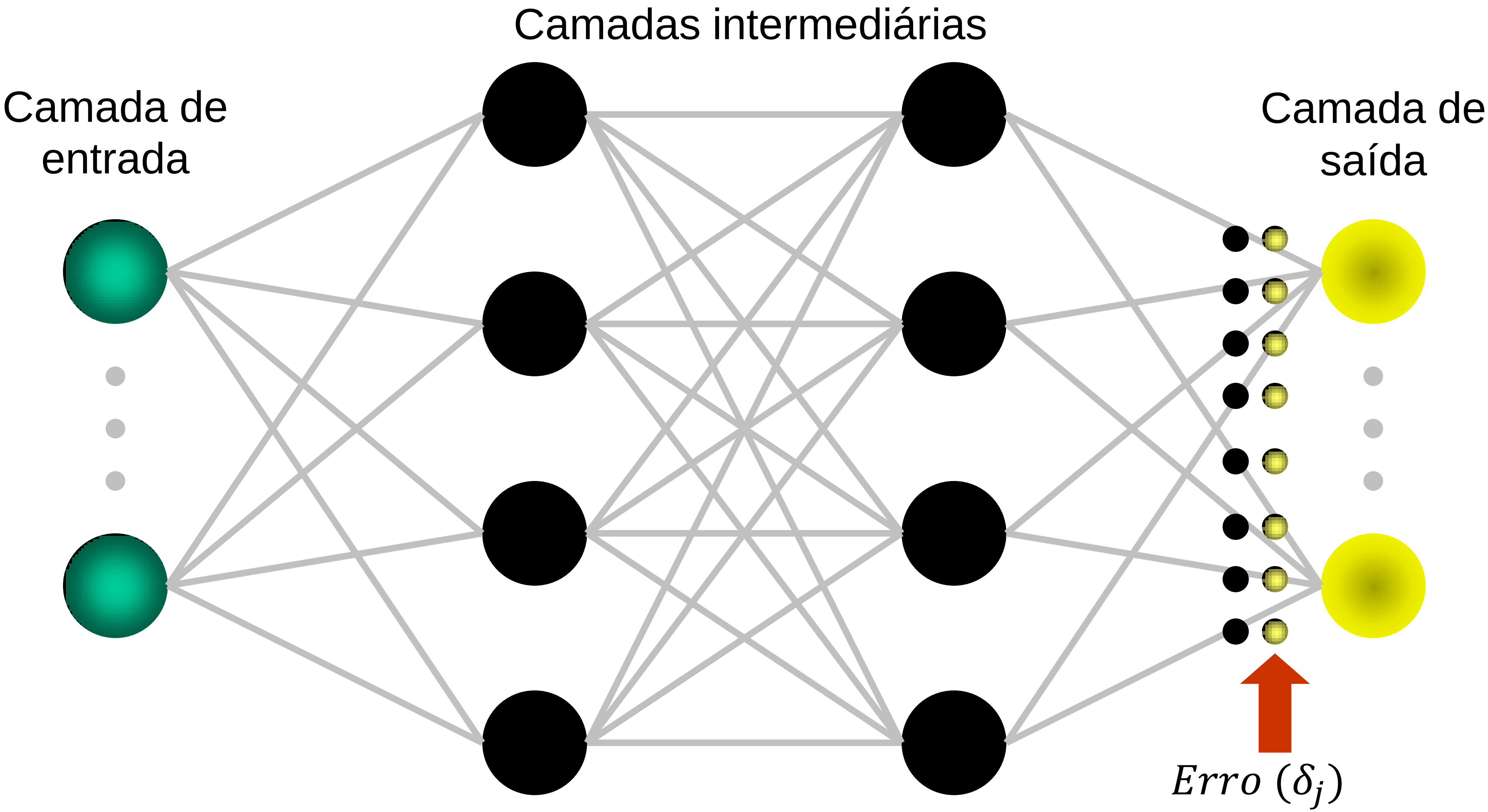
$$\omega_{ji}^{(L)} \leftarrow \omega_{ji}^{(L)} + \eta\delta_j^{(L)} y_i^{(L-1)}$$

$$\omega_{ji}^{(L)} \leftarrow \omega_{ji}^{(L)} + \Delta\omega_{ji}^{(L)}$$

Algoritmo *Backpropagation*, cont.

- **Fase *backward***

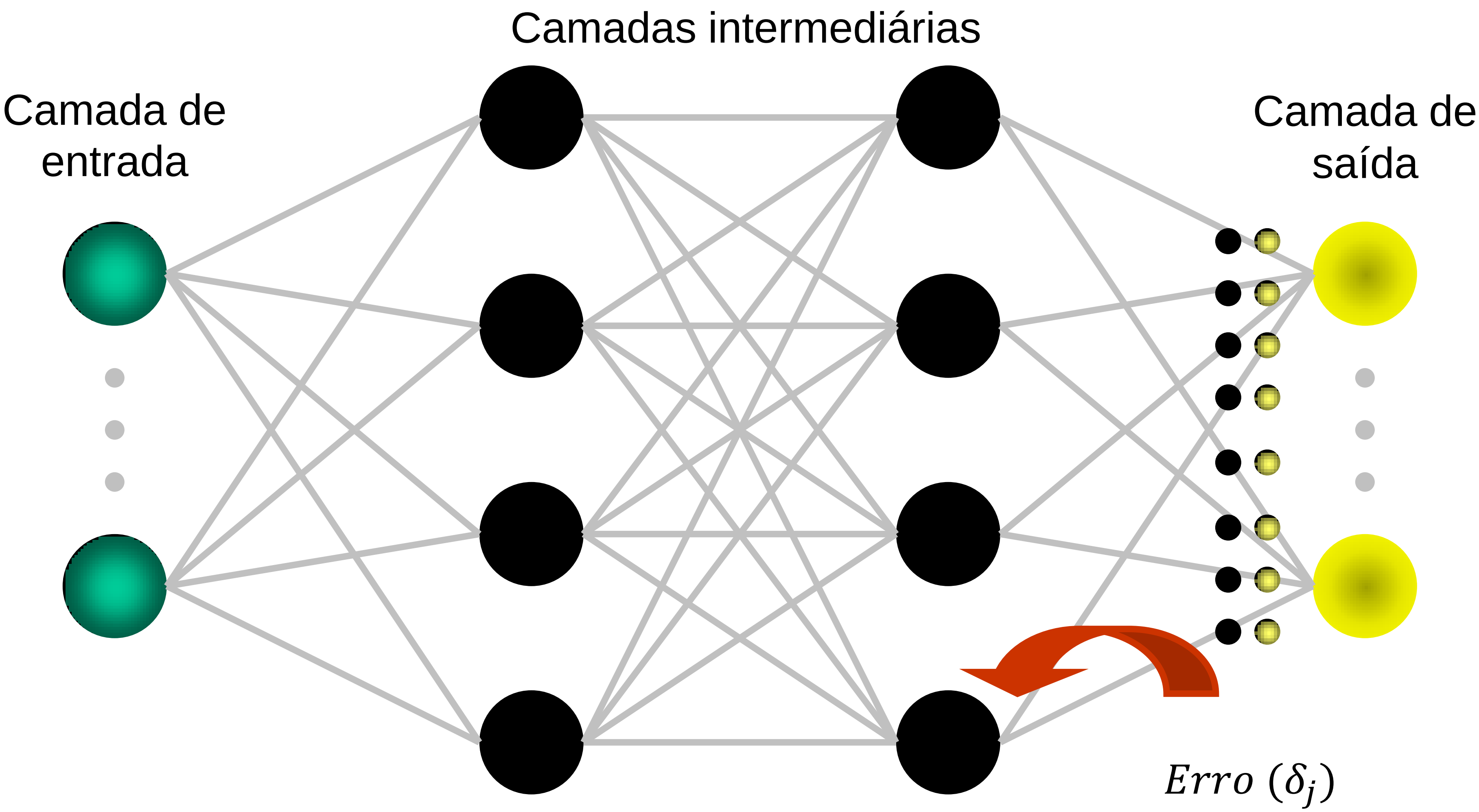
Calcula o termo de correção dos pesos (a atualização será feita depois) $\Delta w_{ji} = \delta_j Y_i$



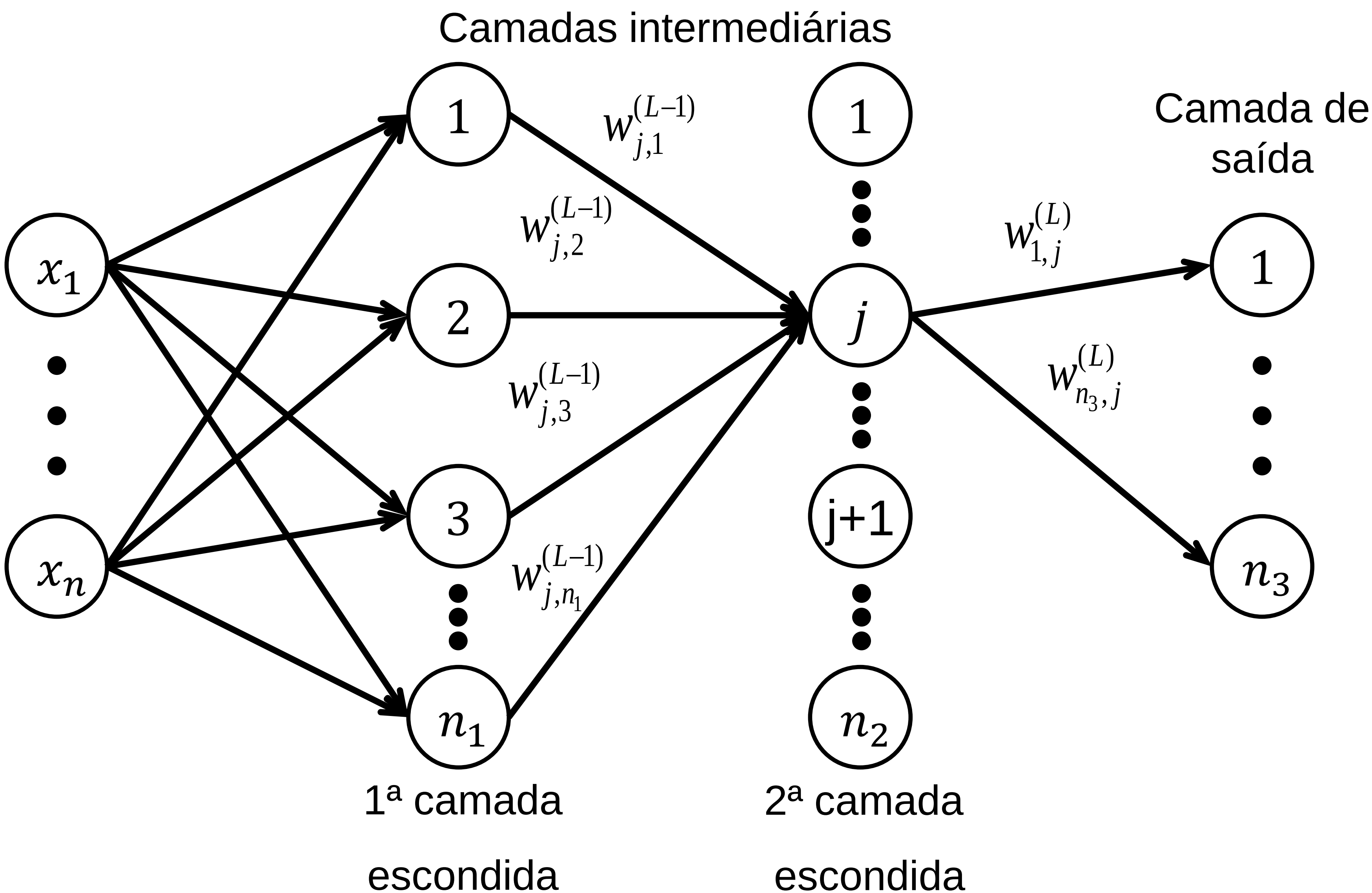
Algoritmo *Backpropagation*, cont.

- Fase *backward*

Propaga o erro para a última camada oculta



Algoritmo *Backpropagation*, cont.



Algoritmo *Backpropagation*, cont.

- Para estes neurônios, não temos um erro $e_j(k)$ explícito.
- Desta forma não temos um gradiente diretamente a partir dos padrões desejados $d_j(k)$.
- Podemos contudo calcular o gradiente local, obtido através da propagação do erro desde a camada de saída.
- Logo:

$$\Delta E^{(L-1)} = \frac{\partial E(k)}{\partial w_{ji}^{(L-1)}(k)} = \frac{\partial E(k)}{\partial y_j^{(L-1)}(k)} \frac{\partial y_j^{(L-1)}(k)}{\partial v_j^{(L-1)}(k)} \frac{\partial v_j^{(L-1)}(k)}{\partial \omega_{ji}^{(L-1)}(k)}$$

Algoritmo *Backpropagation*, cont.

$$v_j^{(L-1)}(k)=\sum_{i=0}^{n_{L-2}}\omega_{ji}^{(L-1)}(k)y_i^{(L-2)}(k)\Rightarrow\frac{\partial v_j^{(L-1)}(k)}{\partial w_{ji}^{(L-1)}(k)}=y_i^{(L-2)}(k)$$

$$y_j^{(L-1)}(k)=\varphi\left(v_j^{(L-1)}(k)\right)\Rightarrow\frac{\partial y_j^{(L-1)}(k)}{\partial v_j^{(L-1)}(k)}=\varphi'\left(v_j^{(L-1)}(k)\right)\frac{\partial v_j^{(L-1)}(k)}{\partial v_j^{(L-1)}(k)}=\varphi'\left(v_j^{(L-1)}(k)\right)$$

$$\frac{\partial E(k)}{\partial y_j^{(L-1)}(k)}=\sum_{l=1}^{n_L}\frac{\partial E(k)}{\partial v_l^{(L)}(k)}\cdot\frac{\partial v_l^{(L)}(k)}{\partial y_j^{(L-1)}(k)}=\sum_{l=1}^{n_L}\frac{\partial E(k)}{\partial v_l^{(L)}(k)}\cdot\frac{\partial(\sum_{l=1}^{n_L}w_{lj}^{(L)}(k)\cdot y_j^{L-1}(k))}{\partial y_j^{(L-1)}(k)}$$

$$\frac{\partial E(k)}{\partial y_j^{(L-1)}(k)}=\sum_{l=1}^{n_L}\frac{\partial E(k)}{\partial v_l^{(L)}(k)}\cdot w_{lj}^{(L)}(k)$$

Neurônios escondidos

- A primeira parte da equação obtida pode ser calculada da seguinte forma:

$$\sum_{l=1}^{n_L} \frac{\partial E(k)}{\partial v_l^{(L)}(k)} = \sum_{l=1}^{n_L} \left(\frac{\partial E(k)}{\partial y_l^{(L)}(k)} \cdot \frac{\partial y_l^{(L)}(k)}{\partial v_l^{(L)}(k)} \right)$$

- Basta agora multiplicar os termos obtidos anteriormente obtendo:

$$\sum_{l=1}^{n_L} \frac{\partial E(k)}{\partial v_l^{(L)}(k)} = \sum_{l=1}^{n_L} \left(-(d_j(k) - y_j^{(L)}(k)) \cdot \varphi'\left(v_j^{(L)}(k)\right)\right) = -\sum_{l=1}^{n_L} \delta_j^{(L)}(k)$$

- Juntando as duas parcelas:

$$\frac{\partial E(k)}{\partial y_j^{(L-1)}(k)} = -\sum_{l=1}^{n_L} \delta_j^{(L)}(k) \cdot w_{lj}^{(L)}(k)$$

Neurônios escondidos

- Finalmente, o gradiente do erro para as camadas intermediárias fica:

$$\frac{\partial E(k)}{\partial w_{ji}^{(L-1)}(k)} = -\left(\sum_{l=1}^{n_L} \delta_j^{(L)}(k)\cdot w_{lj}^{(L)}(k)\right)\cdot \varphi'(v_j^{(L-1)}(k))\cdot y_j^{(L-2)}(k)$$

- O ajuste da matriz de pesos sinápticos pode ser calculada da seguinte forma:

$$\Delta\omega_{ji}^{(L-1)}(k) = -\eta\,\delta_j^{(L-1)}(k)\,y_i^{(L-2)}(k)$$

- Onde

$$\delta_j^{(L-1)}(k) = \sum_{l=1}^{n_L} \delta_j^{(L)}(k)\cdot w_{lj}^{(L)}(k))\cdot \varphi'(v_j^{(L-1)}(k))$$

Observações

- **Passo adiante**: os pesos não se alteram e os sinais são propagados desde a camada de entrada até a saída, neurônio por neurônio. Com isso, podemos calcular os erros $e_j(n)$.

- **Passo retroativo**: começa na camada de saída e caminha no sentido da entrada, passando os sinais de erro e calculando recursivamente os gradientes locais para cada neurônio.

Cálculo de $\varphi'(n)$

- Para a função de ativação do tipo sigmoide:

$$y_j(n) = \varphi_j(v_j(n)) = \frac{1}{1 - e^{-v_j(n)}}$$

$$\varphi_j'(v_j(n)) = \frac{\exp(-v_j(n))}{\left[1 + \exp(-v_j(n))\right]^2} = y_j(n)\left[1 - y_j(n)\right]$$

- Para a função de ativação do tipo tangente hiperbólica:

$$y_j(n) = \varphi_j(v_j(n)) = \tanh\left(a v_j(n)\right) = \frac{e^{a v_j(n)} - e^{-a v_j(n)}}{e^{a v_j(n)} + e^{-a v_j(n)}}$$

$$\varphi_j'(v_j(n)) = a\left[1 - \left(\frac{e^{a v_j(n)} - e^{-a v_j(n)}}{e^{a v_j(n)} + e^{-a v_j(n)}}\right)^2\right] = a\left[1 - y_j^2(n)\right]$$

Fator de aprendizagem η

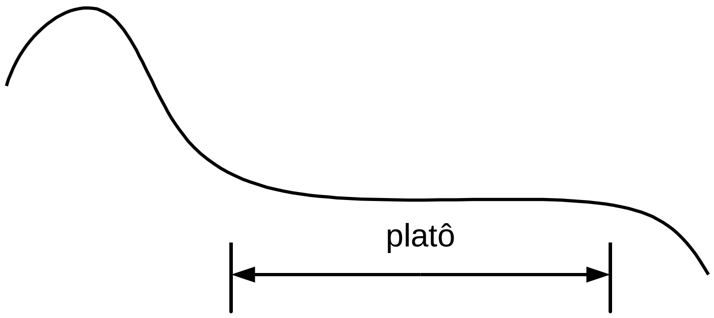
- Normalmente $0.5 \leq \eta \leq 0.8$
- Pode ter um valor diferente para cada camada de pesos sinápticos.
- Deve ter um valor maior nas camadas mais perto da entrada.

Momento

- O momento permite a travessia de platôs com mais facilidade devido à “inércia”.

- Em geral, $0.05 < \alpha < 0.1$

$$\Delta\omega_{ji}(n) = \alpha\Delta\omega_{ji}(n-1) + \eta\delta_j(n)y_i(n)$$



Apresentação dos exemplos para aprendizado

- A ordem de apresentação dos dados durante o aprendizado pode influenciar o desempenho da rede.
- Uma forma de evitar isto é apresentá-los de forma aleatória.
- O conjunto de dados deve ser dividido em duas partes:

  a) aprendizado

  b) testes (validação cruzada, testes finais)
- Se o número total de dados é pequeno, deve-se privilegiar o aprendizado, colocando mais dados neste conjunto

Época de aprendizado

- Sejam ($x_k(n)$, $d_k(n)$), k=1, 2, ...,N, os pares estímulo-resposta para o aprendizado de uma rede neural.

- Uma época de aprendizado se completa quando apresentamos todos os N pares estímulo-resposta ao algoritmo de treinamento.

Atualização dos pesos sinapticos

- **Atualização instantânea:** calcula-se Δw_{ji} e aplica-se a correção a w_{ji} a cada par apresentado.

- **Atualização por lote:** calcula-se os Δw_{ji} para todos os pares, e aplica-se a correção $\sum \Delta w_{ji}$ a w_{ji}, depois que todos os pares foram apresentados.

Observações

- Na atualização instantânea, a estimativa do gradiente não é muito boa, e os passos são dados de forma mais aleatória, enquanto que na atualização por lote temos uma aproximação mais precisa para o gradiente.

- Entretanto, na prática, o primeiro método apresenta resultados em geral melhores (menor probabilidade de parar em mínimos locais).

Inicialização dos pesos sinapticos

- Os w_{ji} iniciais devem ser distribuídos uniformemente no intervalo [-*a*, *a*].

- O parâmetro *a* depende do número de neurônios da camada anterior.

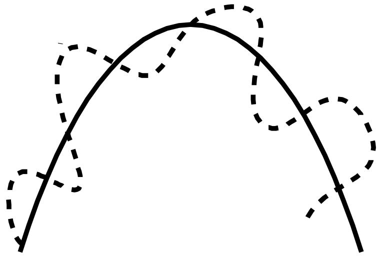
- Pode ser ajustado verificando-se a saturação de $\varphi'(.)$ (dependendo do número de entradas do neurônio).

Underfitting e Overfitting

- Estes erros são relativos à razão entre o número de neurônios e o número de dados de treinamento.

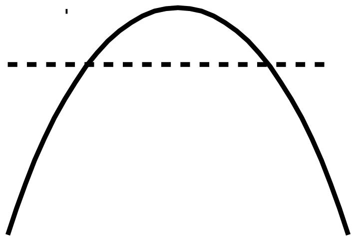
Overfitting

✓ erros pequenos no treinamento, mas erros grandes no teste



Underfitting

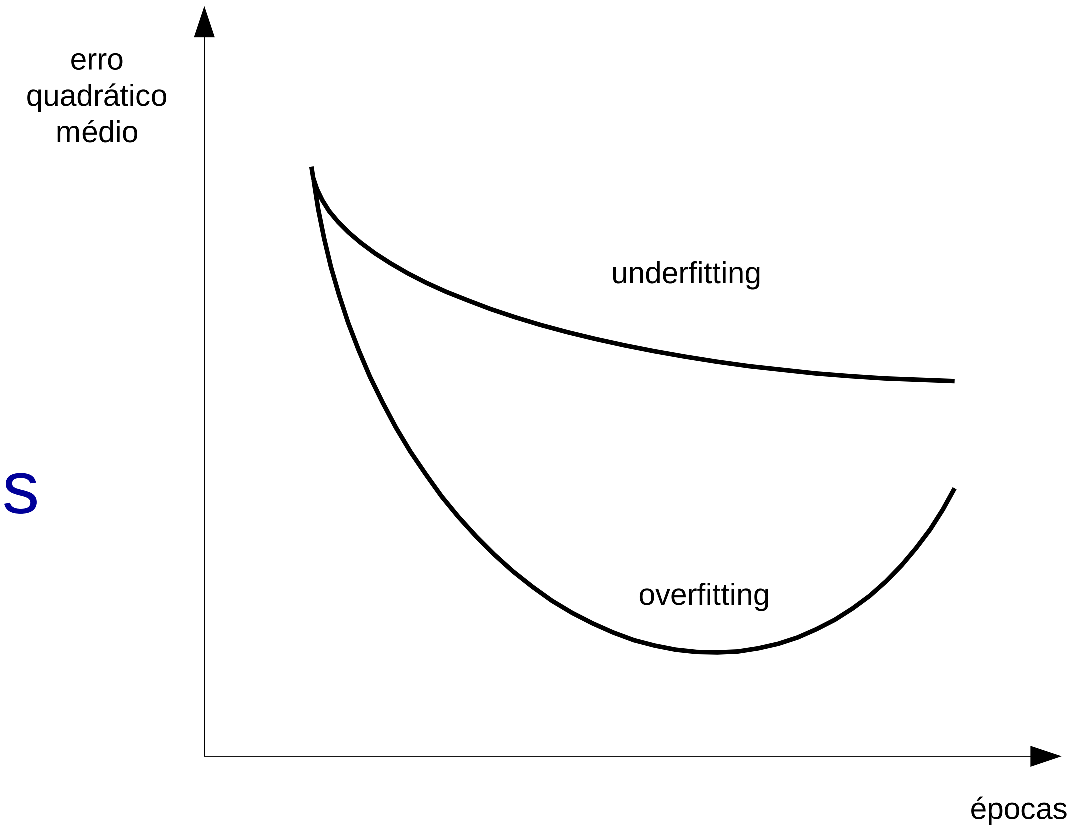
✓ erros grandes no treinamento e no teste



Underfitting e Overfitting

- Número de exemplos para o treinamento (N)

$$N > \frac{W}{E}$$



W: número de pesos

E: erro admissível

Validação Cruzada

- Consiste em fazer a verificação da convergência da rede através de um conjunto de dados diferentes daqueles usados para o aprendizado.
- Tenta evitar que a rede fique viciada nos dados de aprendizado.
- Com isto, dividimos a nossa base de dados em 3 partes:
  - ✓ pares de aprendizado
  - ✓ pares de validação cruzada
  - ✓ pares de teste

EPC-4

- Realizar programa para o treinamento e uso de rede neural com duas camadas sinápticas e função de ativação tangente hiperbólica, com o algoritmo backpropagation, para a seguinte função:
- Sejam {d1,d2,d3} três dígitos diferentes e

$$g(x_1, x_2, x_3) = \frac{1}{3}\left[\mathrm{sen}(d_1 x_1) + \mathrm{sen}(d_2 x_2) + \mathrm{sen}(d_3 x_3)\right], \quad -\frac{\pi}{2} \leq d_i x_i \leq \frac{\pi}{2}$$

- Treine a rede para reproduzir g e durante o treinamento, plote o erro quadrático médio.
- Utilize 10000 pontos do R3 no cubo dado pelos intervalos $d_i x_i$, gerados aleatoriamente com distribuição uniforme, para testar a rede treinada.
- Calcule o valor do erro quadrático médio e o maior erro quadrático observado no teste.

Fim

Edielson Prevato Frigieri
edielson@inatel.br