

OUTLINE - PART I

1. MLP Neural Networks
2. Backpropagation Algorithm

MULTILAYER PERCEPTRON

- Redes de apenas uma camada só representam funções linearmente separáveis:
- Problema XOR (**limitação**)
- Redes de múltiplas camadas solucionam essa restrição
- O desenvolvimento do algoritmo Back-Propagation dos motivos para o ressurgimento da área de rede

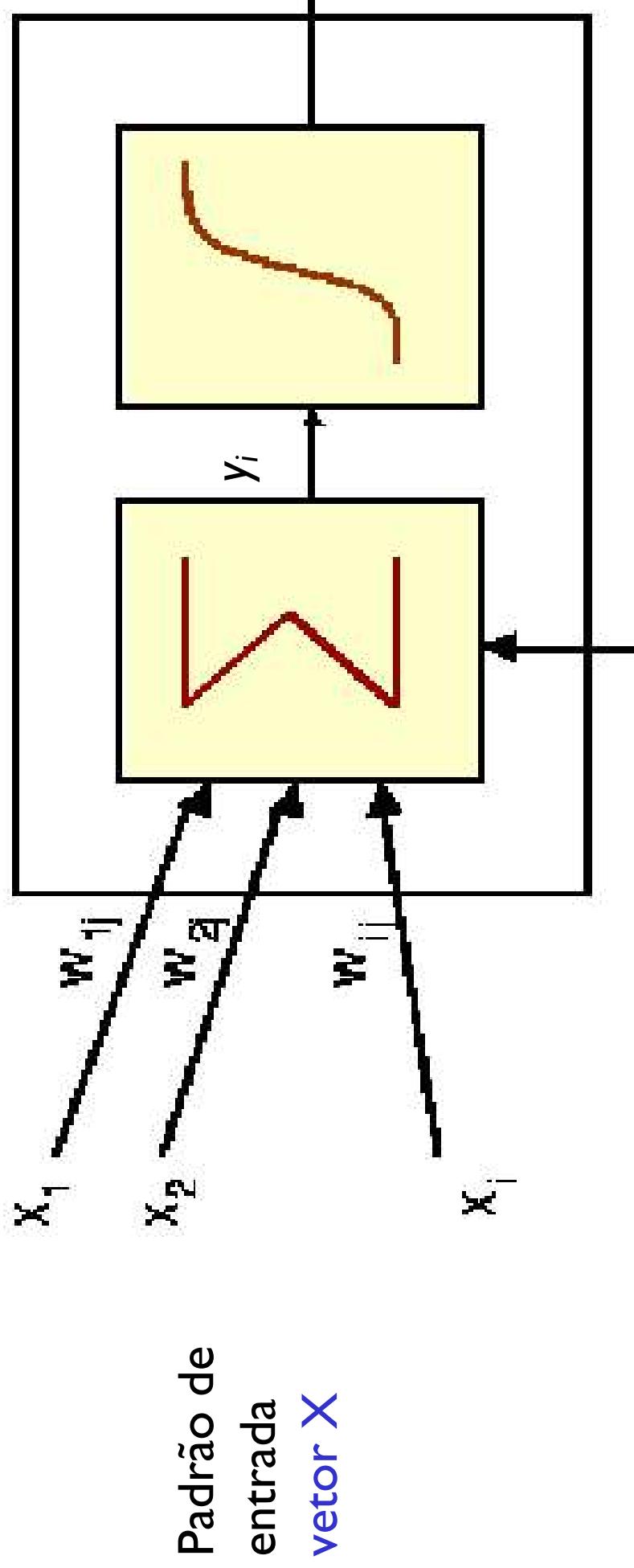
MULTILAYER PERCEPTRON

- O grande desafio foi achar um algoritmo de aprendizado para atualizar os pesos das camadas intermeio

MULTILAYER PERCEPTRON

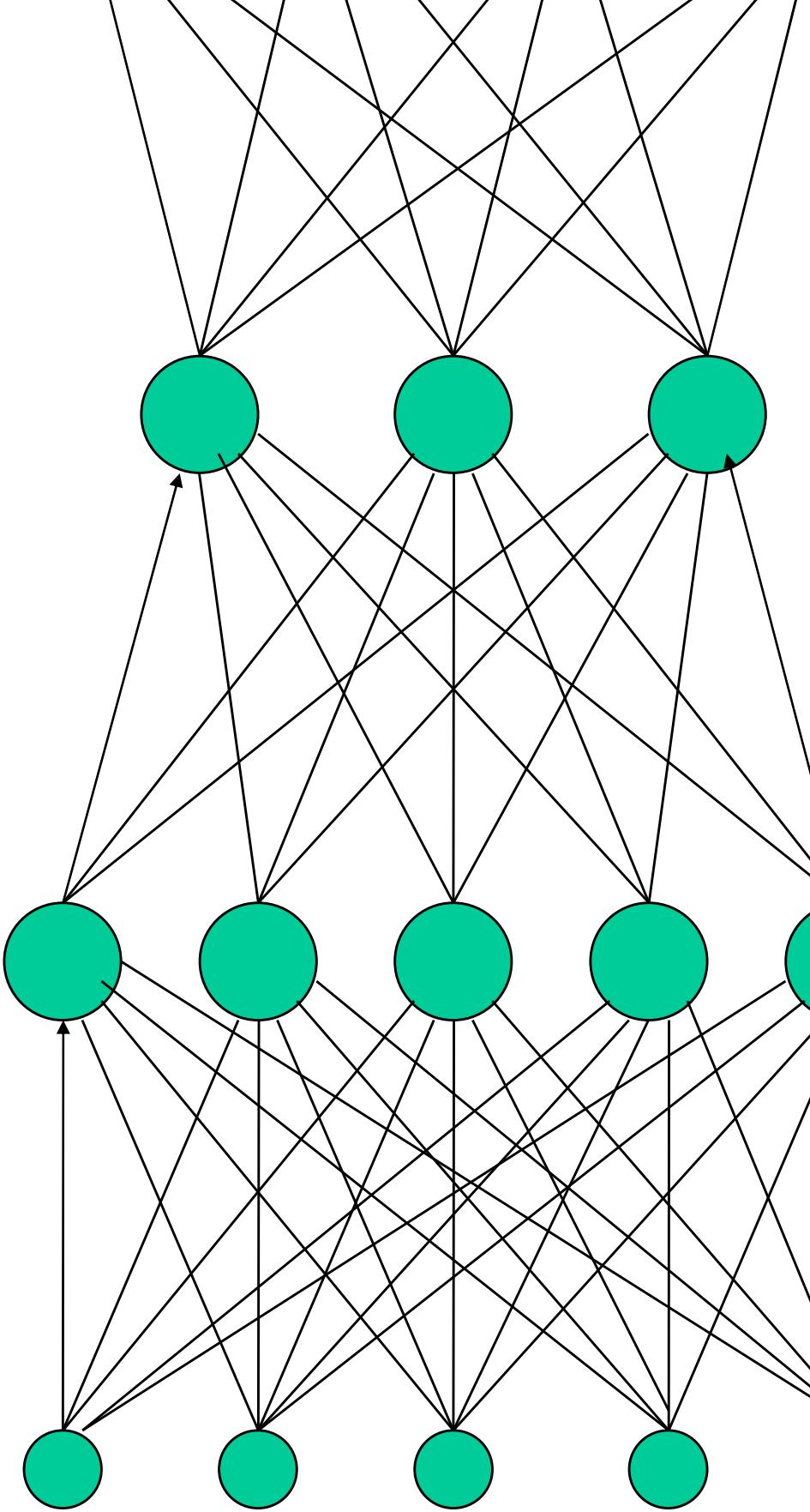
- ▶ O grande desafio foi achar um algoritmo de atualizar dos pesos das camadas intermeios
- ▶ Idéia Central:
 - ▶ os erros dos elementos processadores da saída (conhecidos pelo treinamento supervisionado) são propagados para as camadas intermeios

MULTILAYER PERCEPTRON - MLP



MULTILAYER PERCEPTRON - ARCHITECTURE

MLP with 3 layers Input + 5/3/4



MULTILAYER PERCEPTRON

- Regra de propagação
- Função de ativação: Função não-linear diferenciável em todos os pontos [$s_i = F(y_i)$] (**is that true?**)
- Topologia: Múltiplas camadas
- Algoritmo de Aprendizado: Supervisionado
- Valor de Entrada/Saída: Binários e/ou Contínuos

MLP - LEARNING

- ▶ Processo de minimização do erro quadrático pelo
Gradiente Descendente

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

- ▶ Cada peso sináptico i do elemento processador j proporcionalmente ao negativo da derivada parcial deste processador com relação ao peso.

MLP - LEARNING

- No qual o erro quadrático do processador j é definido
- $E_j = \frac{1}{2} (t_j - s_j)^2$
- t_j – valor desejado de saída para o processador j da camada saída
- s_j – estado de ativação do processador j da camada

MLP - LEARNING

- Na verdade, deve-se minimizar o erro de todos os pontos da camada de saída:

$$E_p = \frac{1}{2} \sum_{j=1}^N (t_j - s_j)^2$$

- t_j – valor desejado de saída do padrão p para o processador j da camada de saída
- s_j – estado de ativação do processador j da camada de saída

MLP - LEARNING

- Cálculo Δw_{ij}

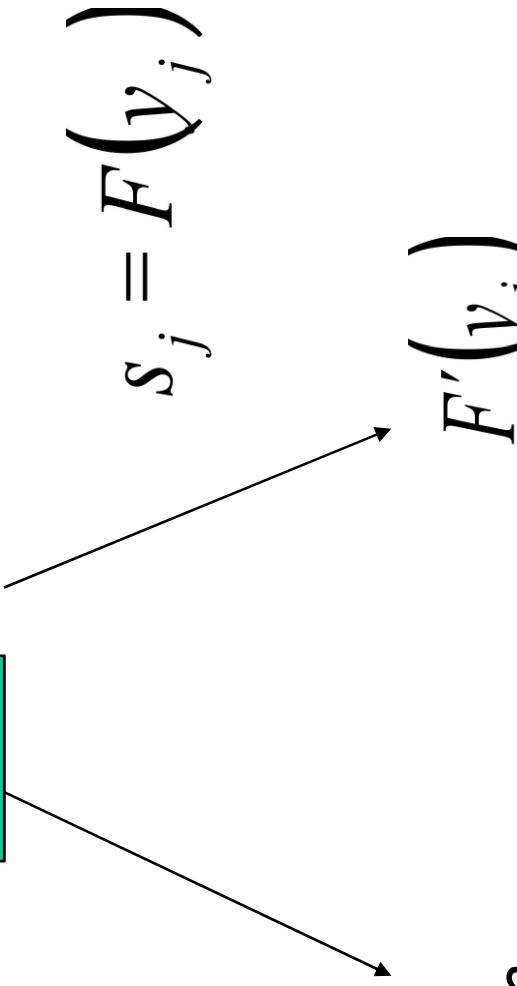
$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} = -\eta \frac{\partial E_p}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}}$$

$y_j = \sum x_i w_i$

MLP - LEARNING

- Depende da camada à qual o processador j pertence

$$\delta_j = -\frac{\partial E_p}{\partial y_j} = \boxed{-\frac{\partial E_p}{\partial s_j} \frac{\partial s_j}{\partial y_j}}$$



Se i = Camada de Saída

MLP - LEARNING (OUTPUT NEU)

$$\delta_j = -\frac{\partial E_p}{\partial y_j} = \boxed{-\frac{\partial E_p}{\partial s_j} \frac{\partial s_j}{\partial y_j}}$$

$$E_p = \frac{1}{2} \sum (t_j - s_j)^2$$

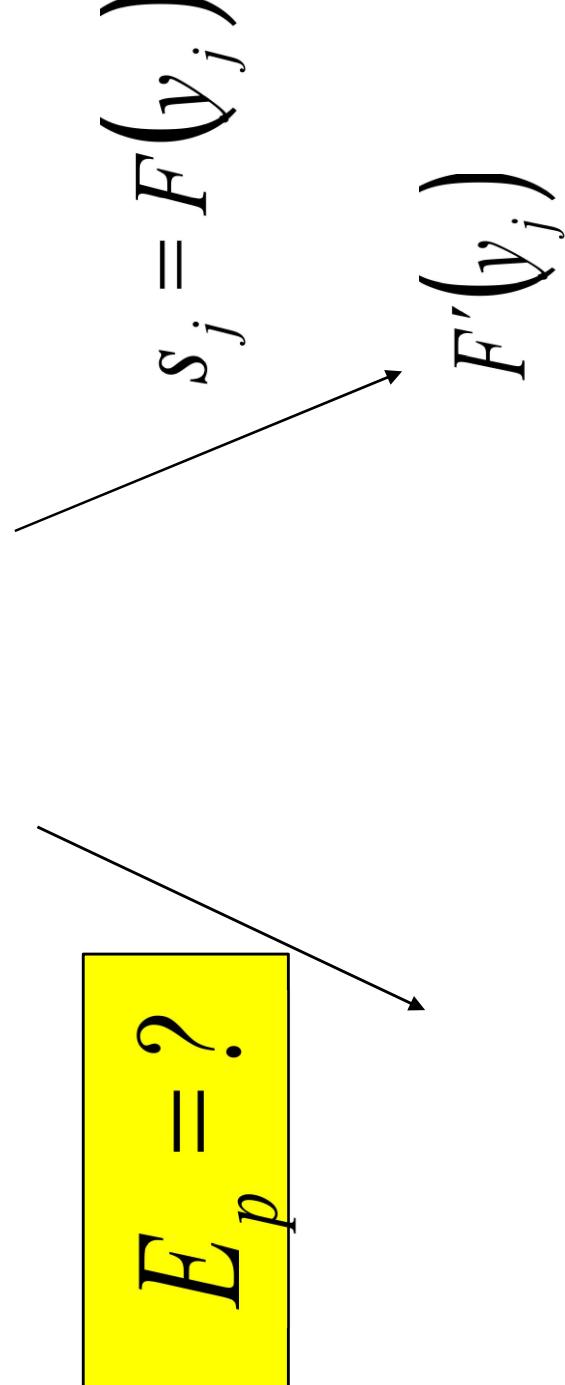
$$s_j = F(y_j)$$

$$F'(y_j)$$

$$2 \times \frac{1}{2} \times (t_j - s_j) \chi(-1)$$

MLP - LEARNING (HIDDEN NEU)

$$\delta_j = -\frac{\partial E_p}{\partial y_j} = \boxed{\frac{\partial E_p}{\partial s_j} \frac{\partial s_j}{\partial y_j}}$$

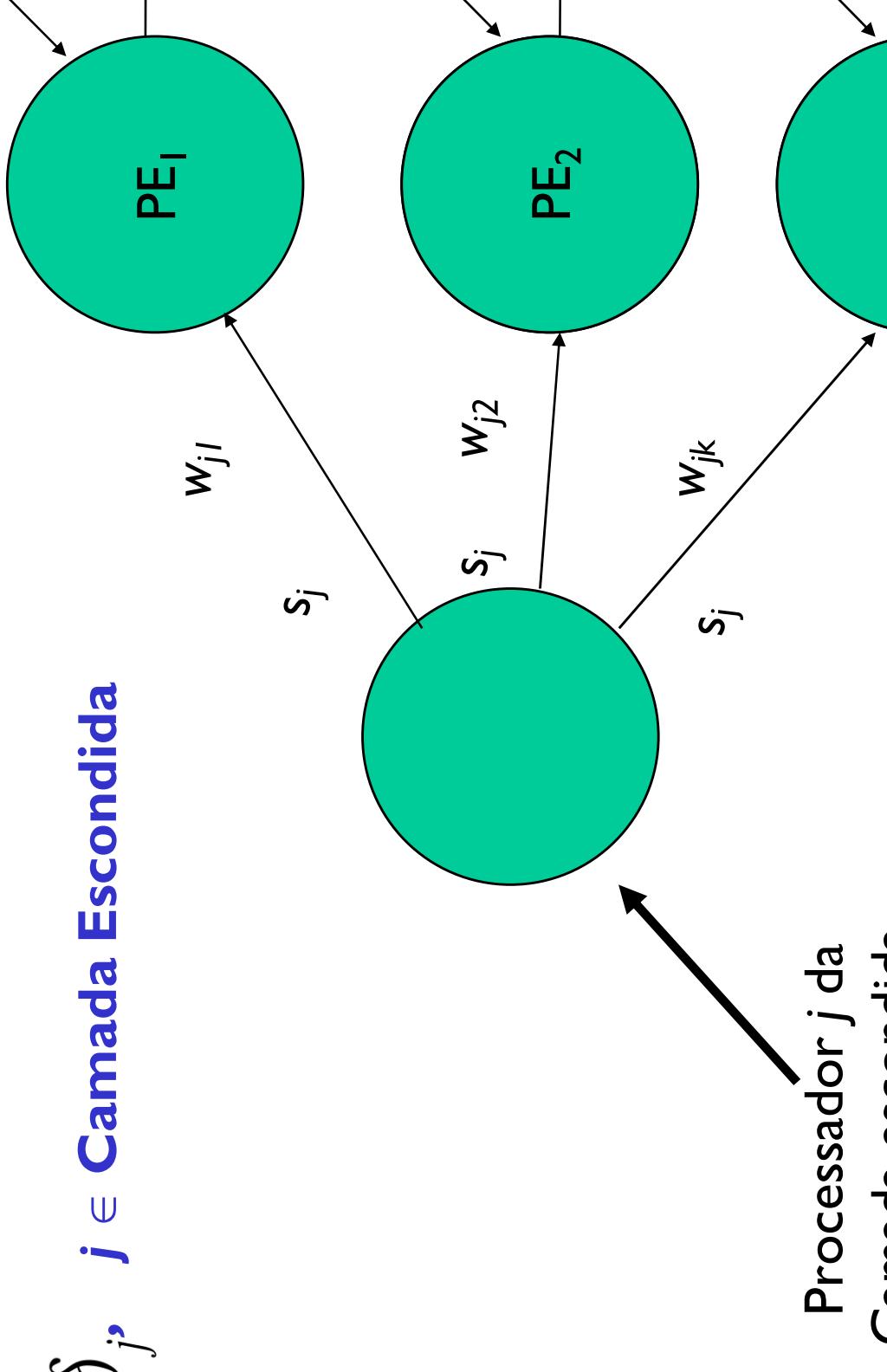


MLP - LEARNING (HIDDEN NEU

- Pelo aprendizado supervisionado, só se conhece o erro saída;
- Erro na saída é função do potencial interno do processo;
- O y_k depende dos estados de ativação dos processadores da camada anterior ($s_j = x_j$) e dos pesos das conexões (w_{jk});
- Portanto, s_j de uma camada escondida afeta, em maior grau, o erro de todos os processadores da camada subsequente.

MLP - LEARNING (HIDDEN NEU)

Cálculo de δ_j , $j \in$ Camada Escondida



$$\delta_j = -\frac{\partial E_p}{\partial y_j} = -\frac{\partial E_p}{\partial s_j} \frac{\partial s_j}{\partial y_j}$$

$$= -\sum_k e_k \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial s_j}$$

Temos que $e_k = t_k - s_k = t_k - F(y_k)$

$$= -F'(y_j) \sum_k e_k \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial s_j}$$

$$y_k = \sum_j w_{jk} s_j$$

$$\frac{\partial y_k}{\partial s_j} = w_{jk}$$

Logo:

Campo Local Induzido do Neurônio k
Derivando em relação a s_j , obtemos:

$$\delta_j = -F'(y_j) \sum_k e_k \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial s_i}$$

MLP - LEARNING

- In summary:

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

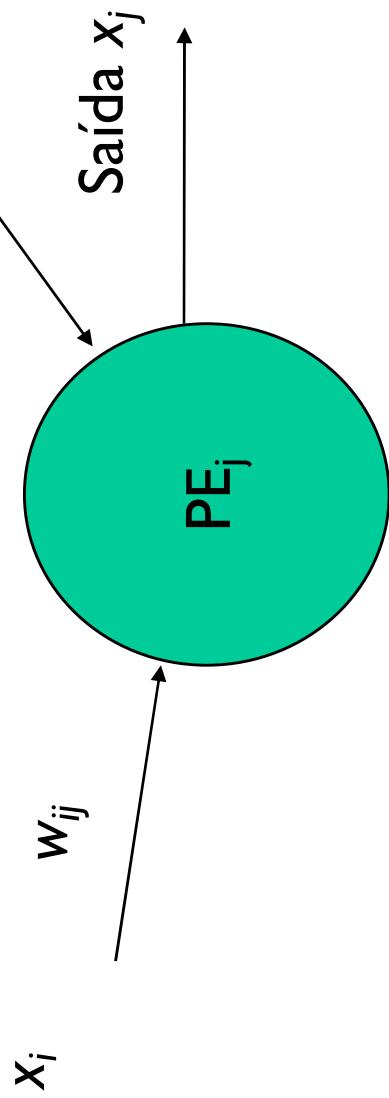
Onde

x_i – valor de entrada recebido pela conexão i
 δ_j – valor calculado do erro do processador j

MLP - LEARNING

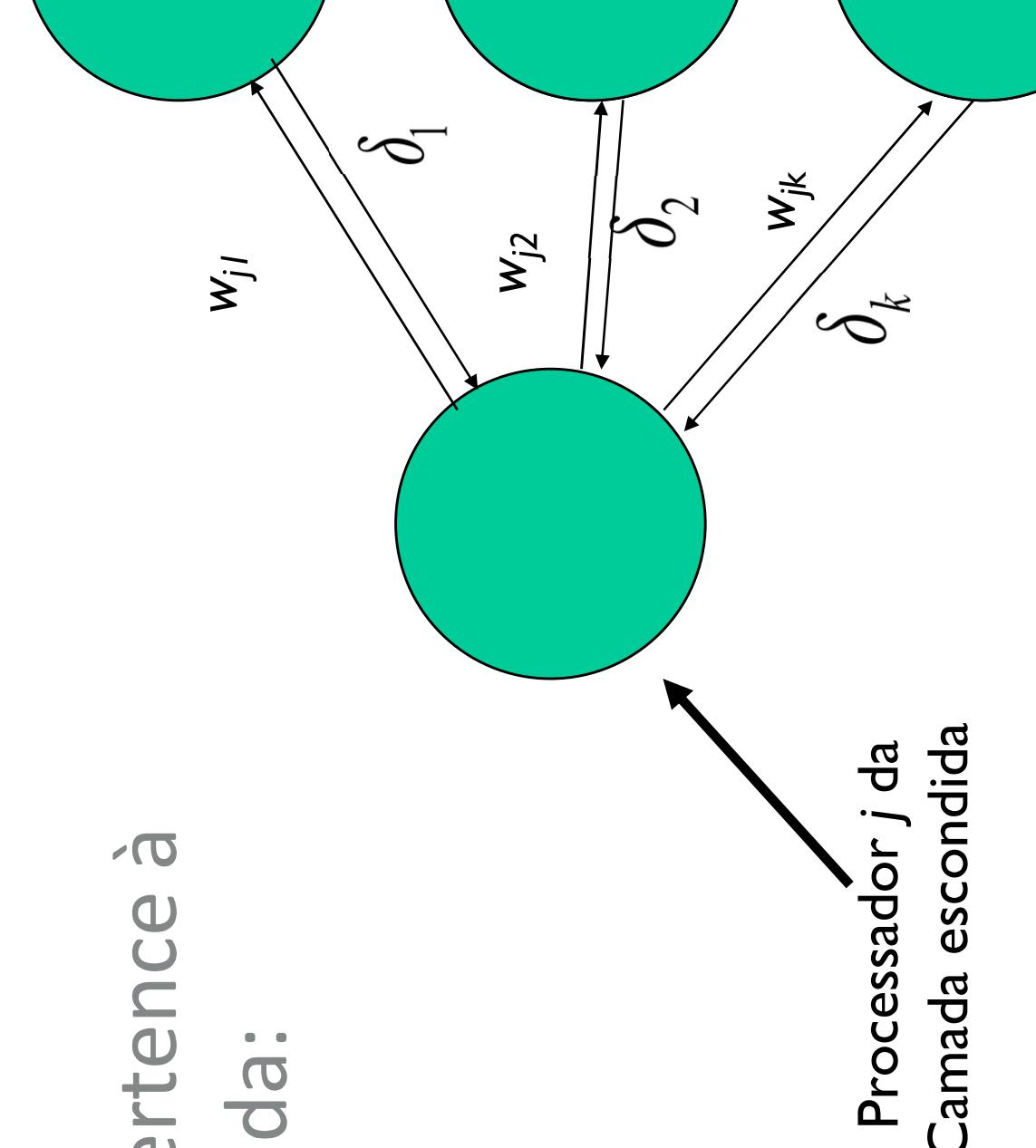
Valor desejado
de saída t_j

Processador j pertence à Camada de Saída:



MLP - LEARNING

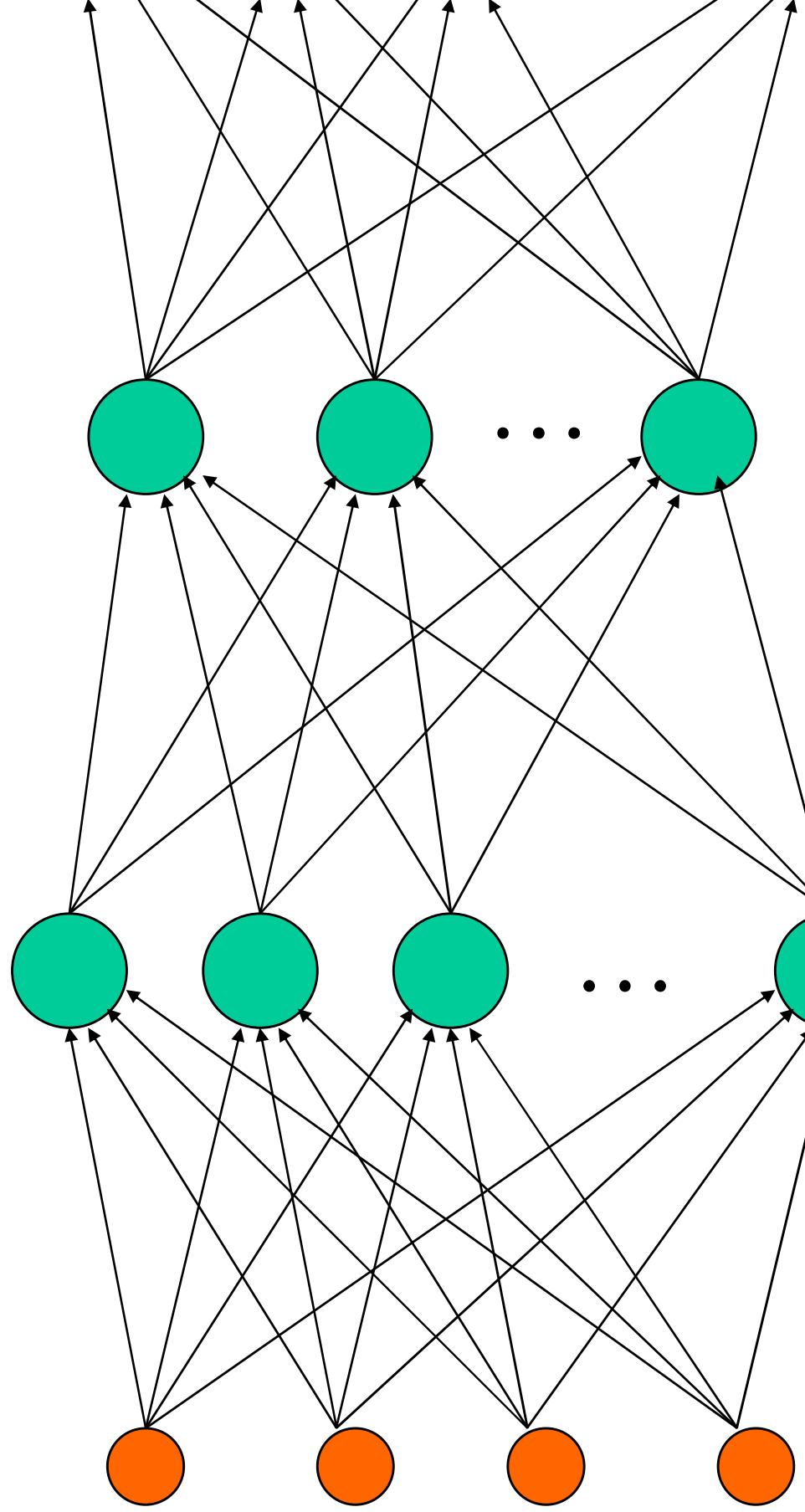
- ▶ Processador j pertence à Camada Escondida:



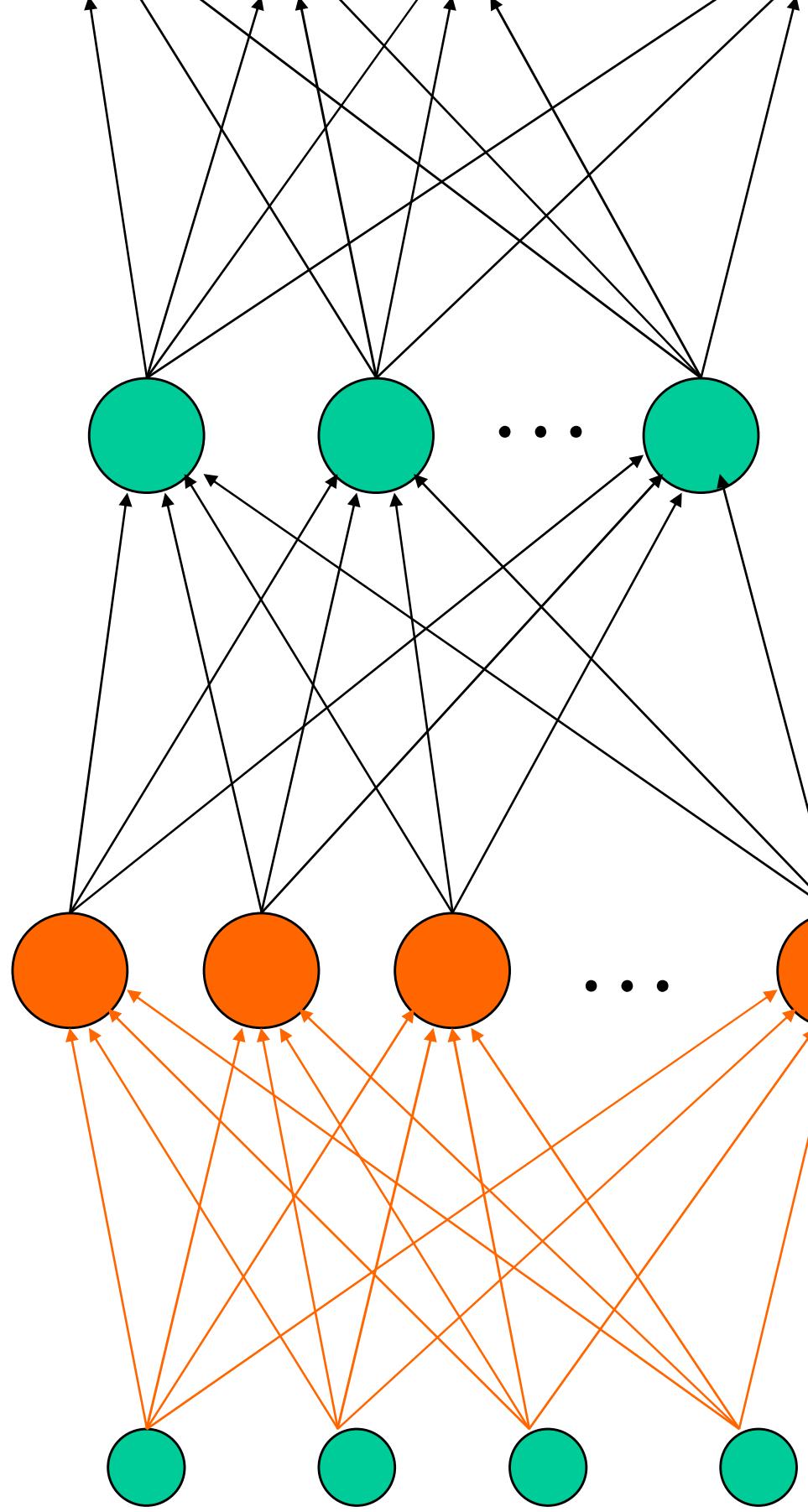
MLP - LEARNING

- ▶ O algoritmo Back-Propagation tem portanto duas etapas para cada padrão apresentado
- ▶ **Feedforward** - as entradas se propagam, pelas camadas de entrada para a camada de saída
- ▶ **Feedback** - os erros se propagam, na direção das camadas de saída ao fluxo de dados, ou seja, da camada de saída para a camada escondida

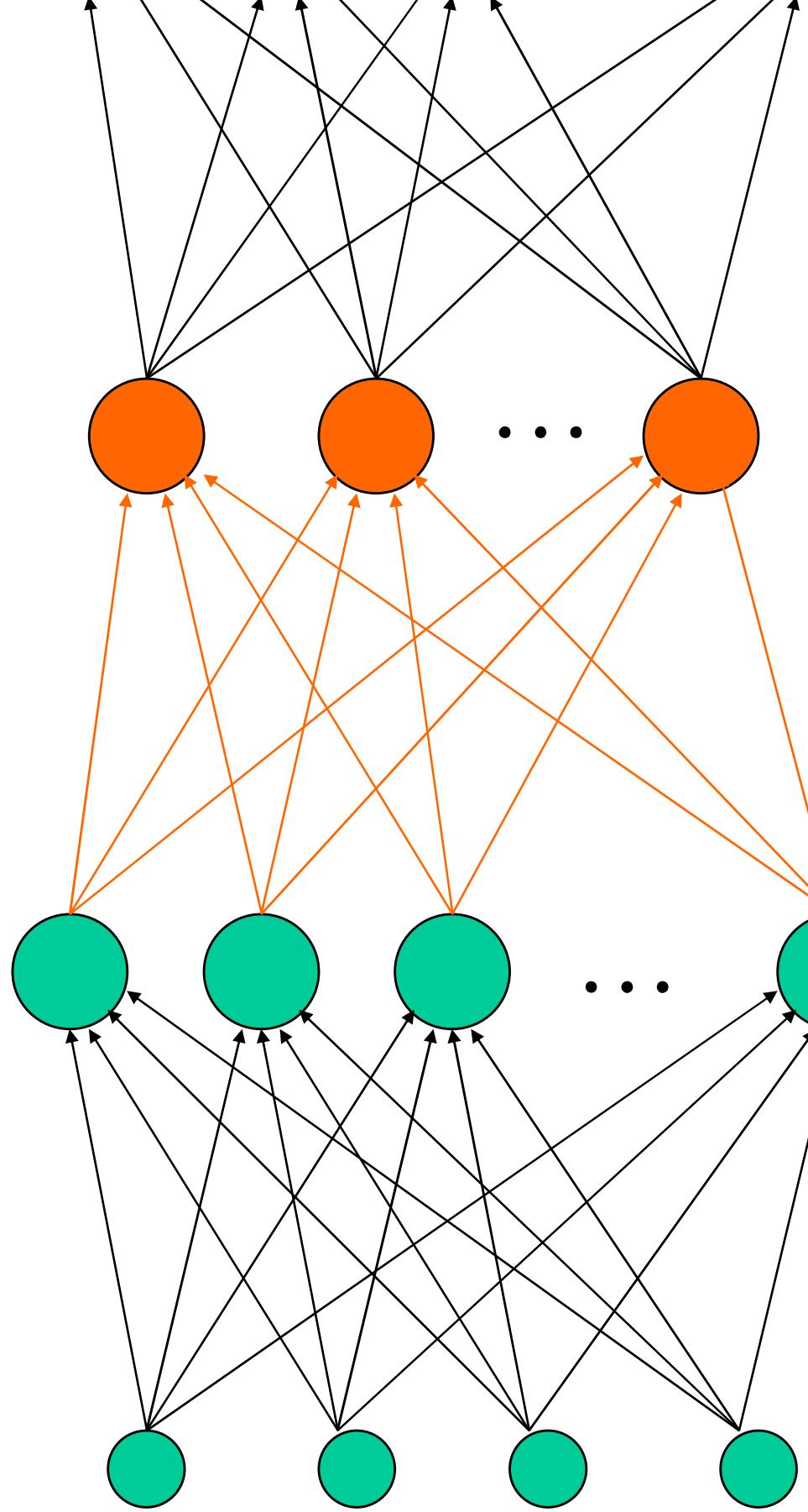
ILLUSTRATION



ILLUSTRATION



ILLUSTRATION



ILLUSTRATION

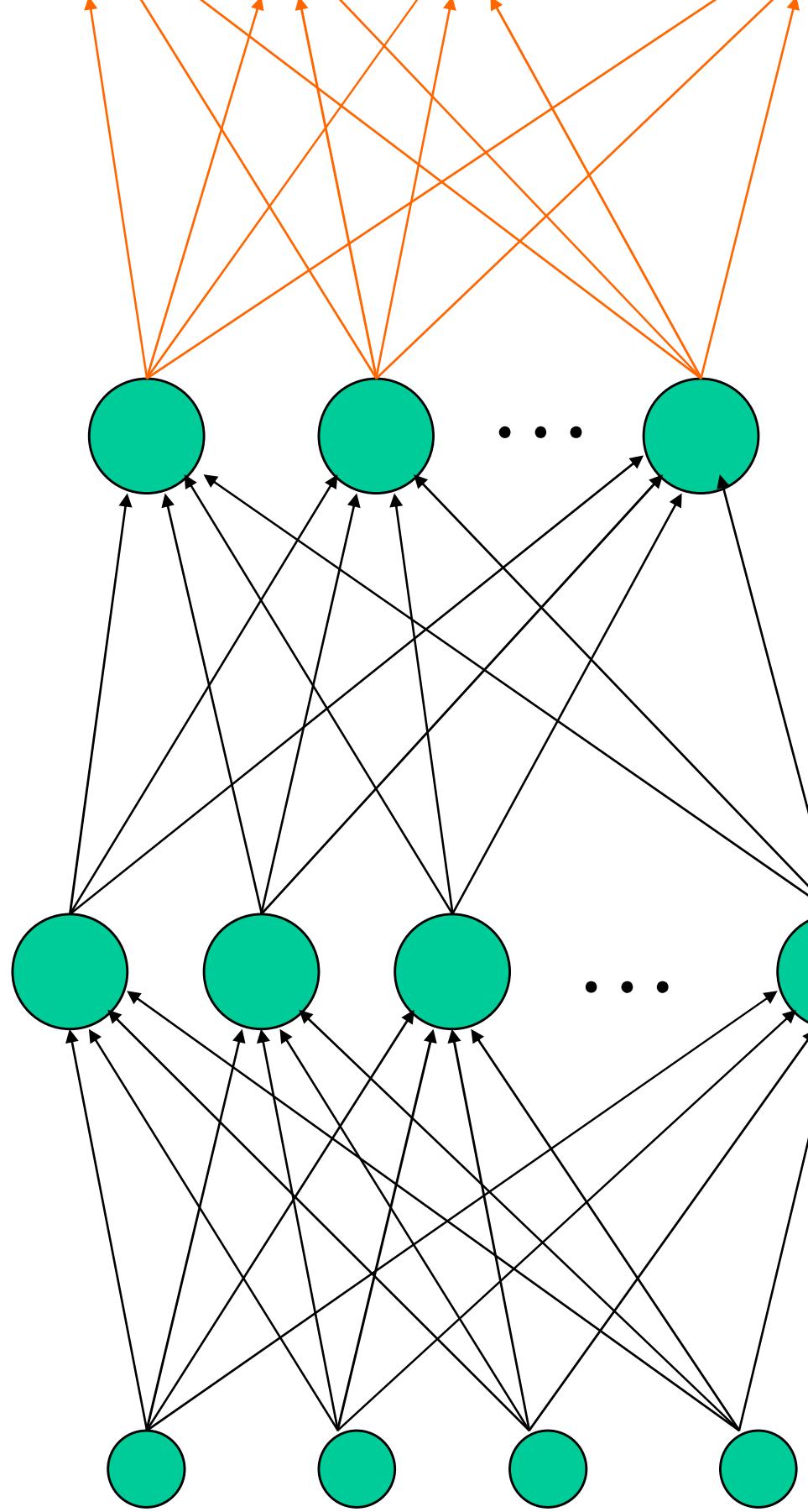


ILLUSTRATION - BACKWARD

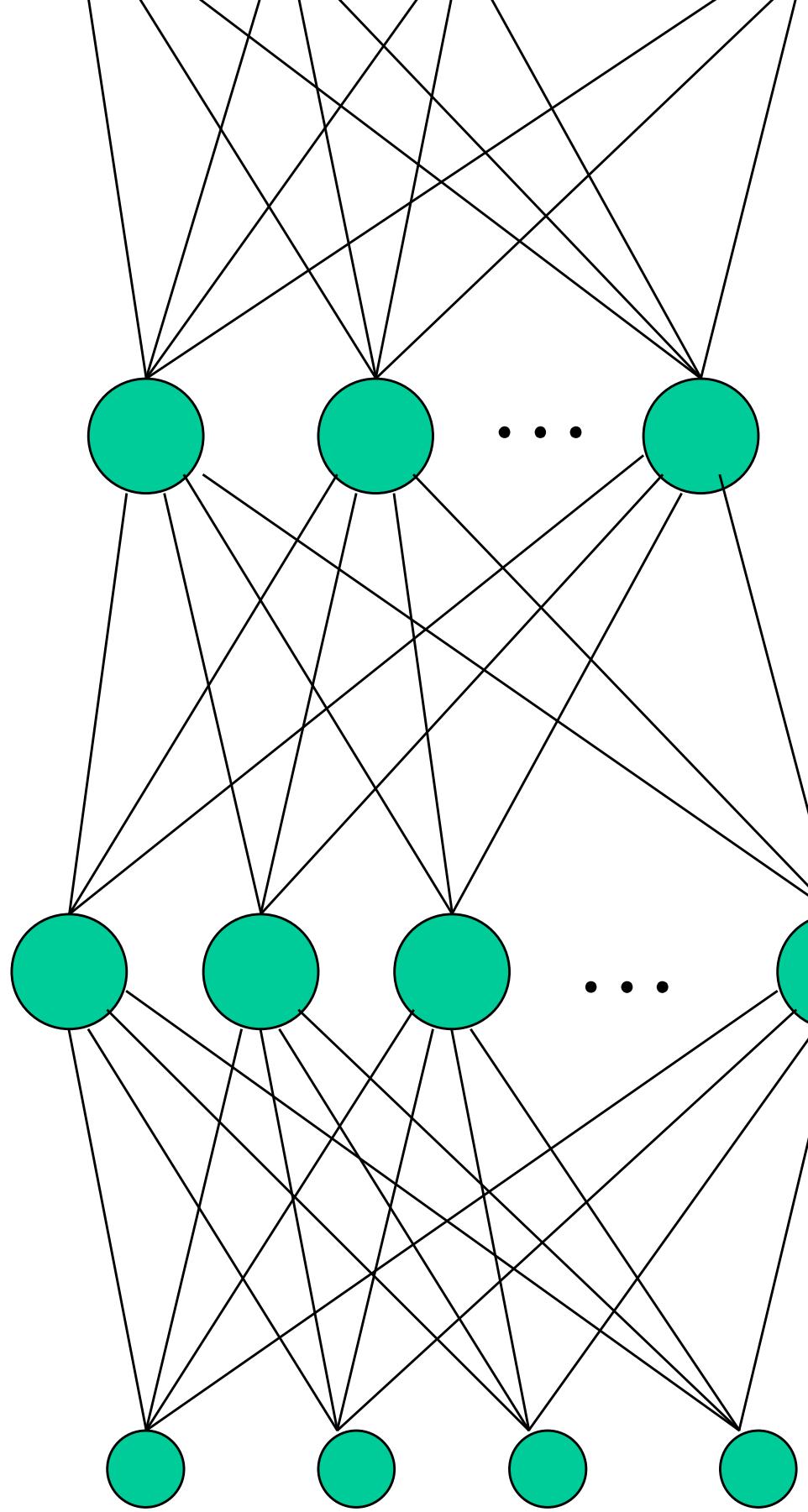


ILLUSTRATION - BACKWARD

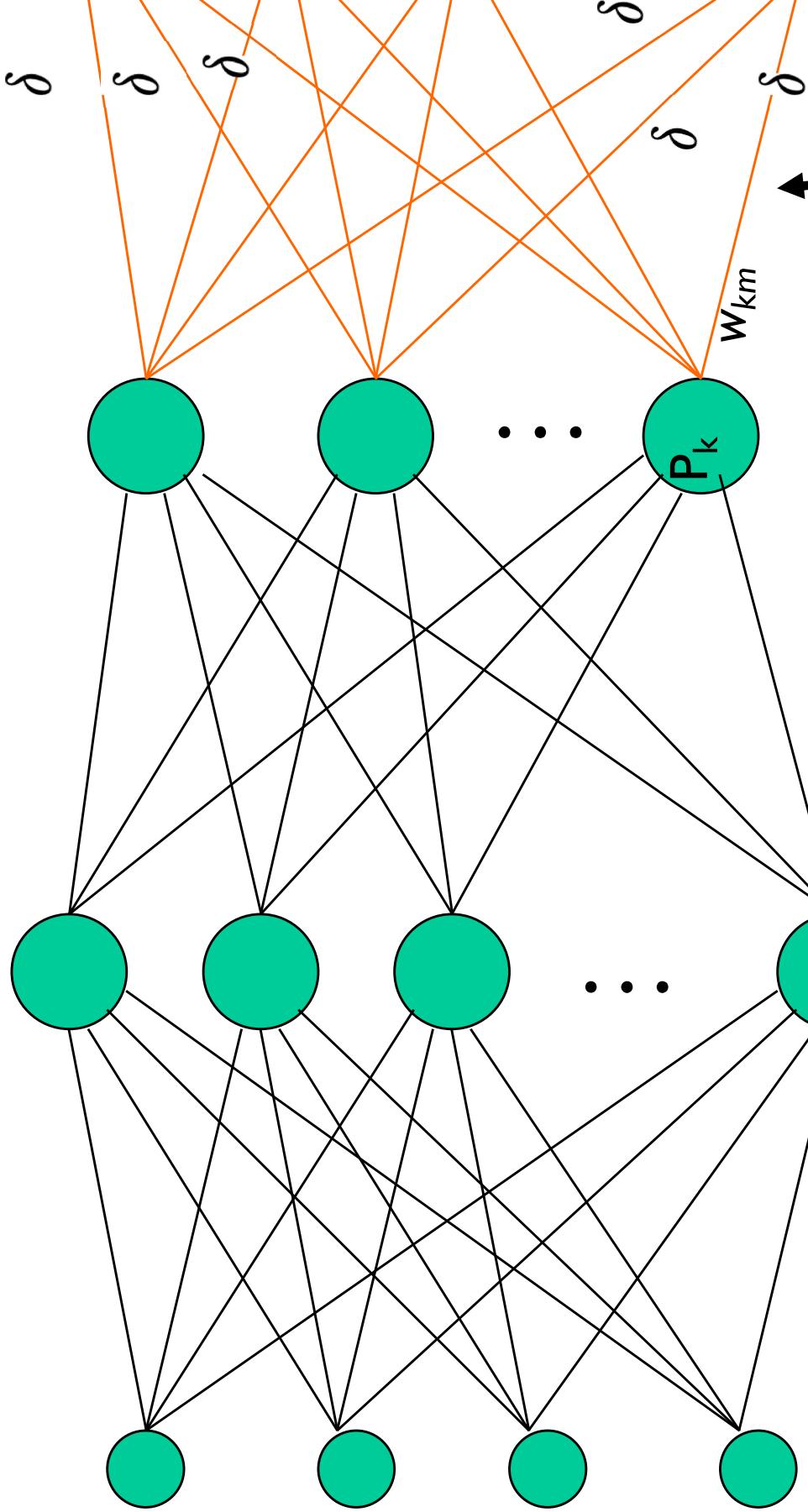


ILLUSTRATION - BACKWARD

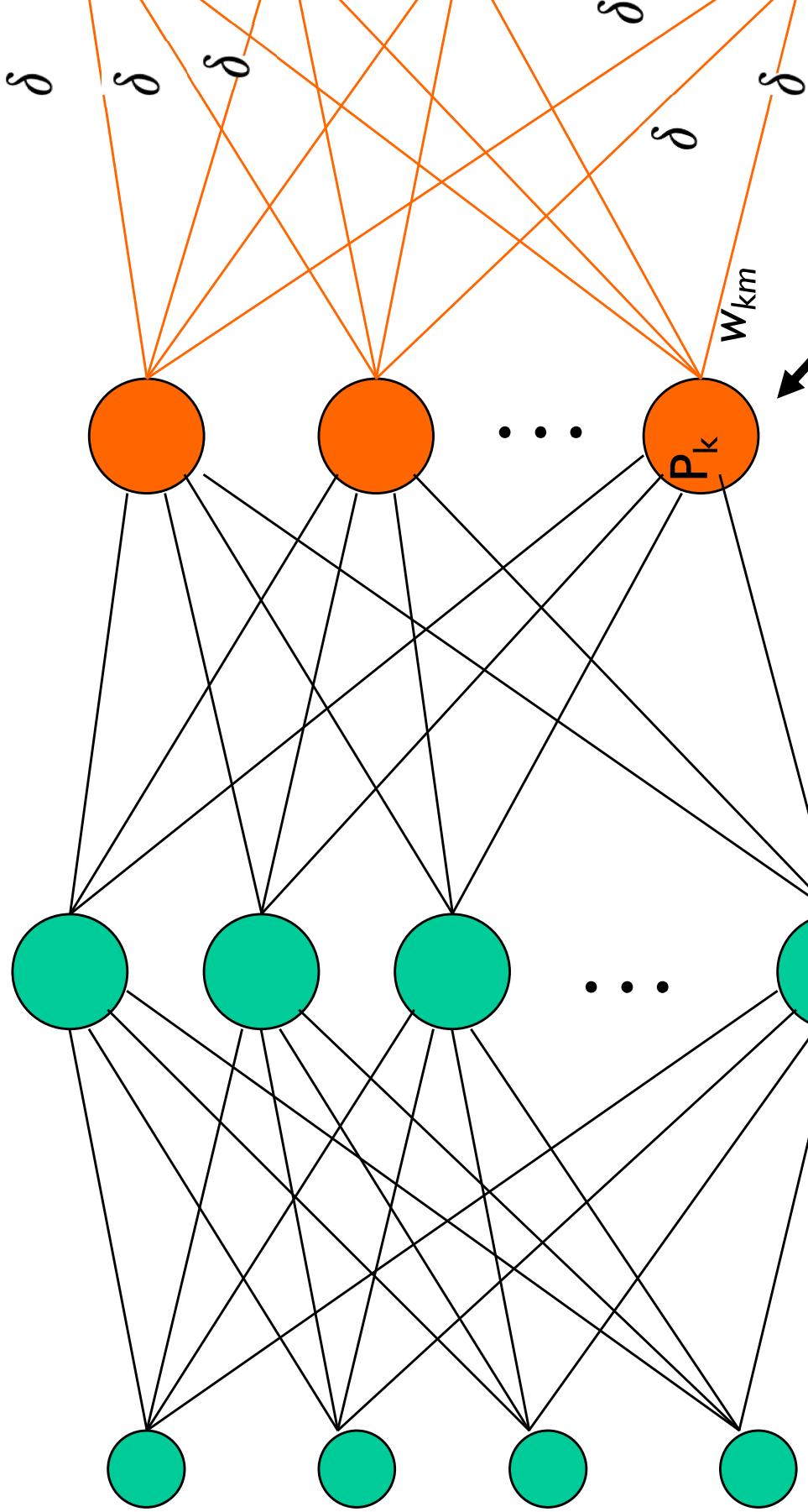


ILLUSTRATION - BACKWARD

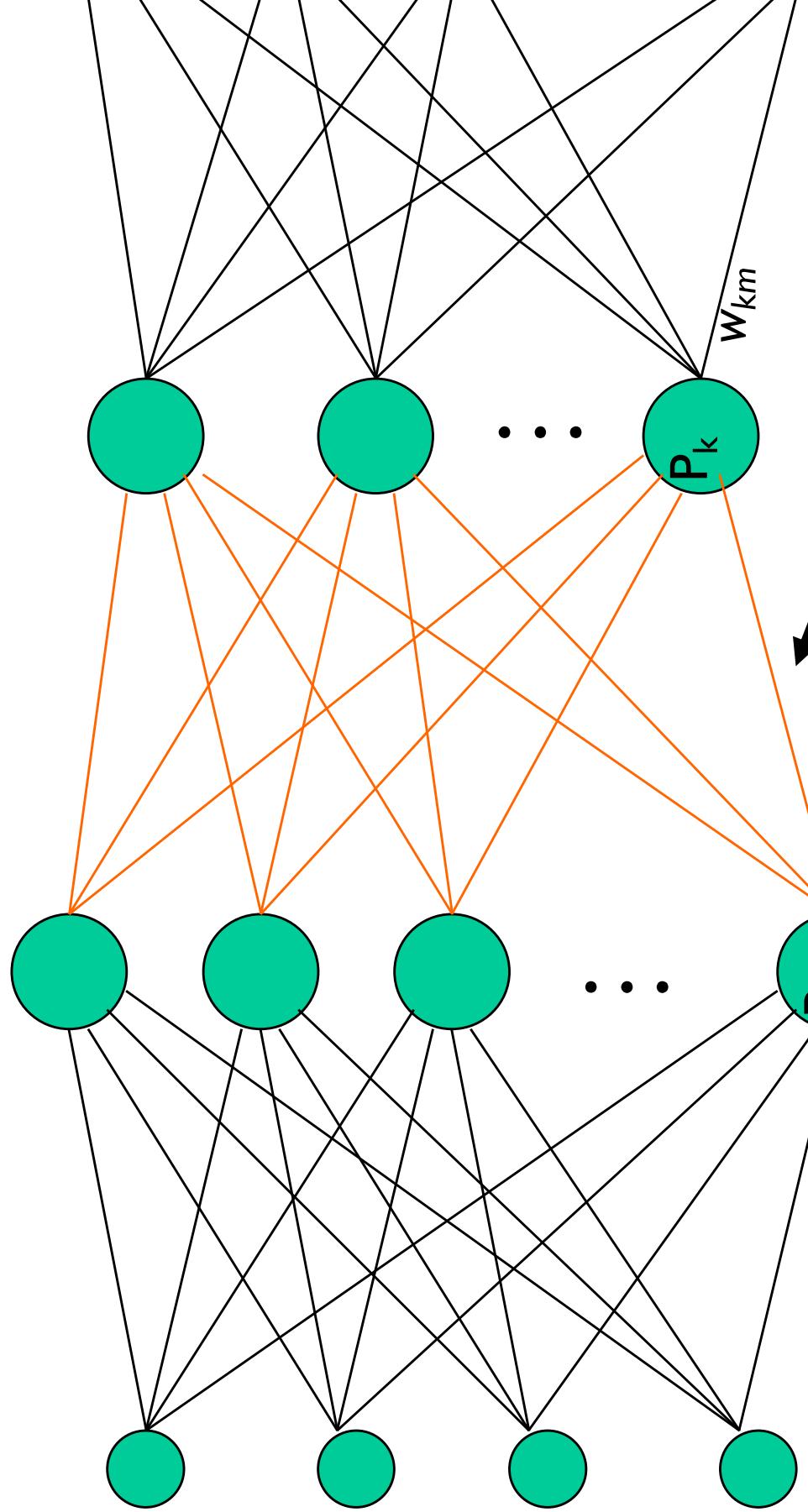


ILLUSTRATION - BACKWARD

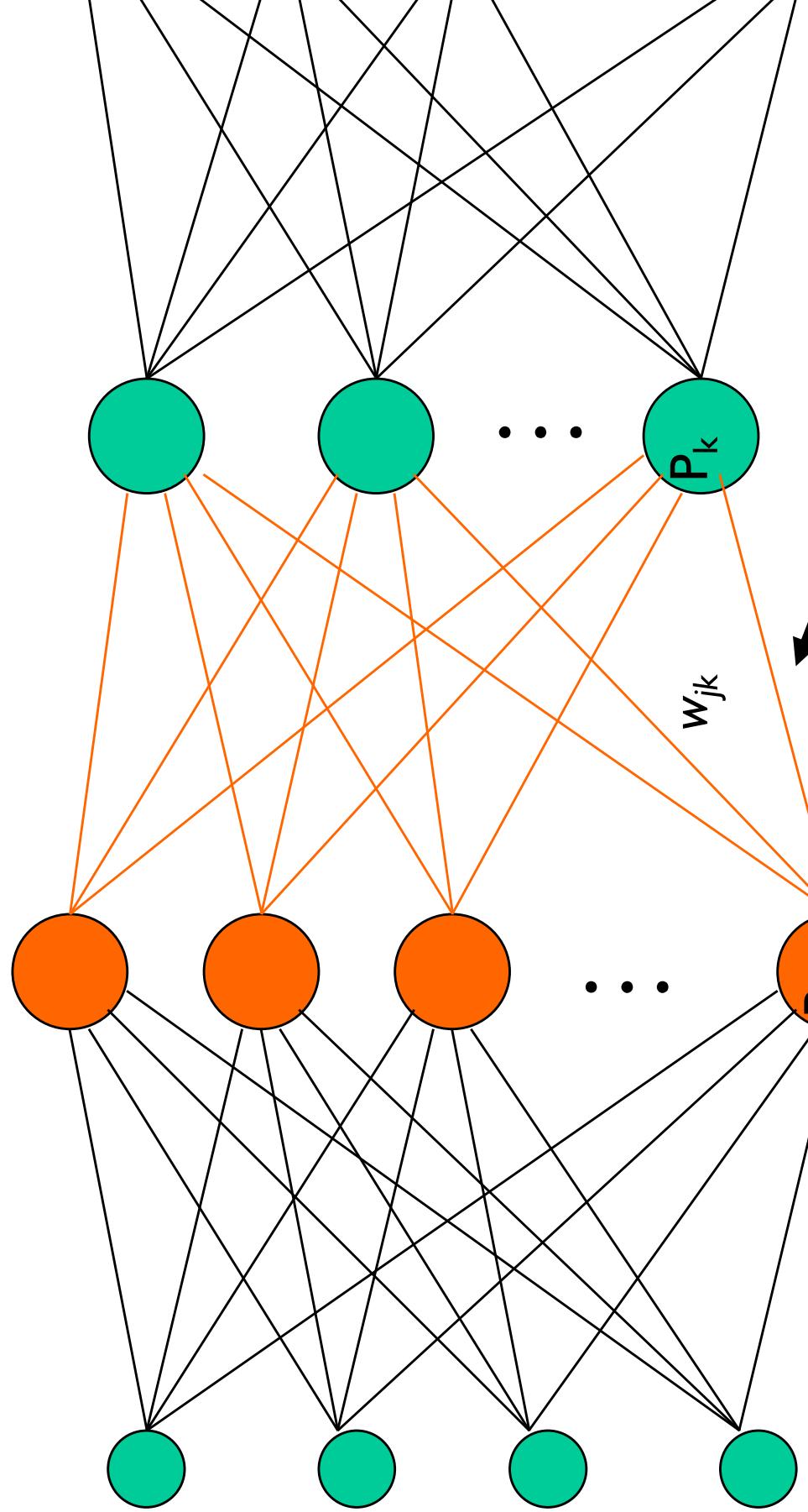
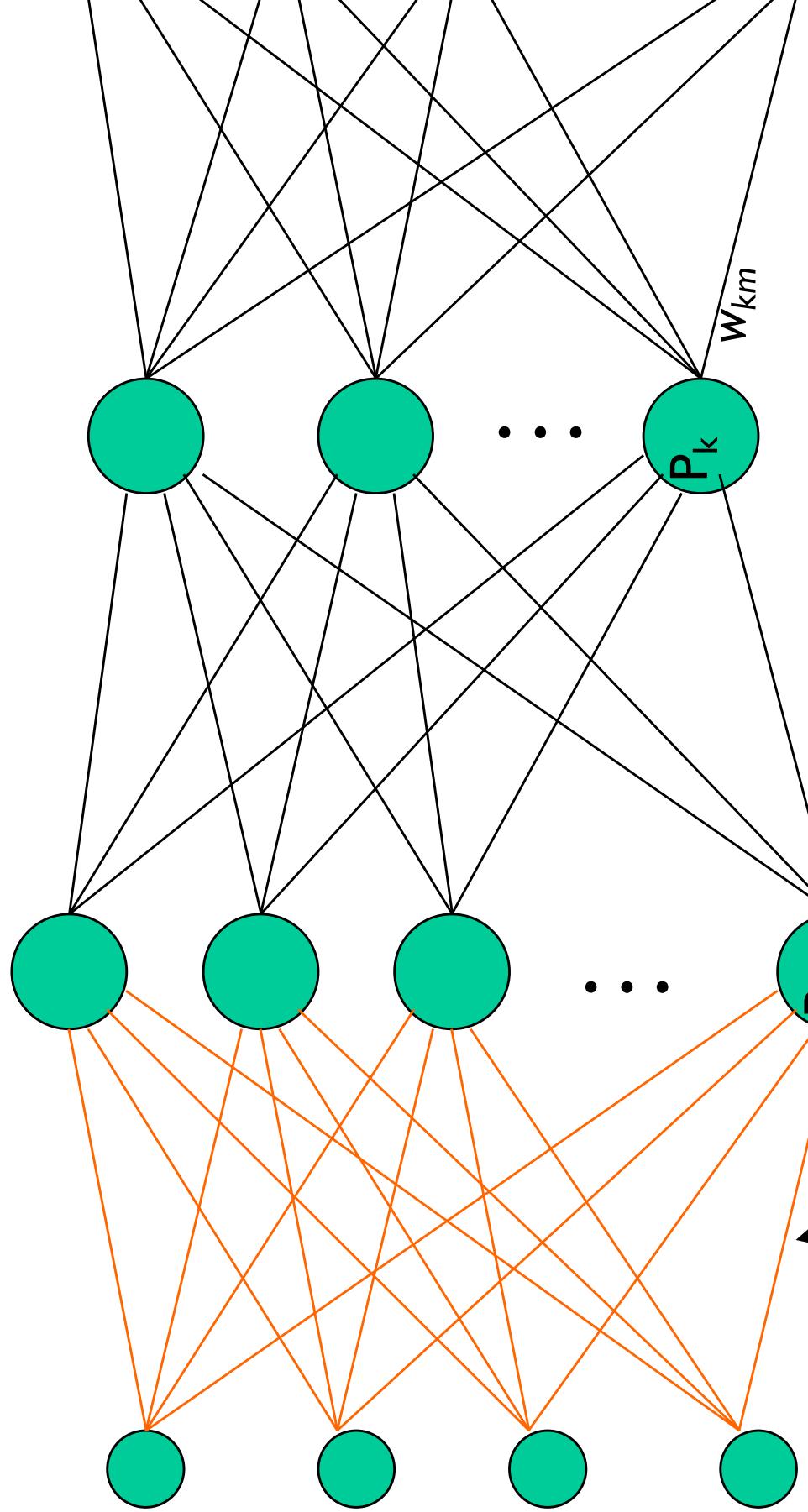


ILLUSTRATION - BACKWARD

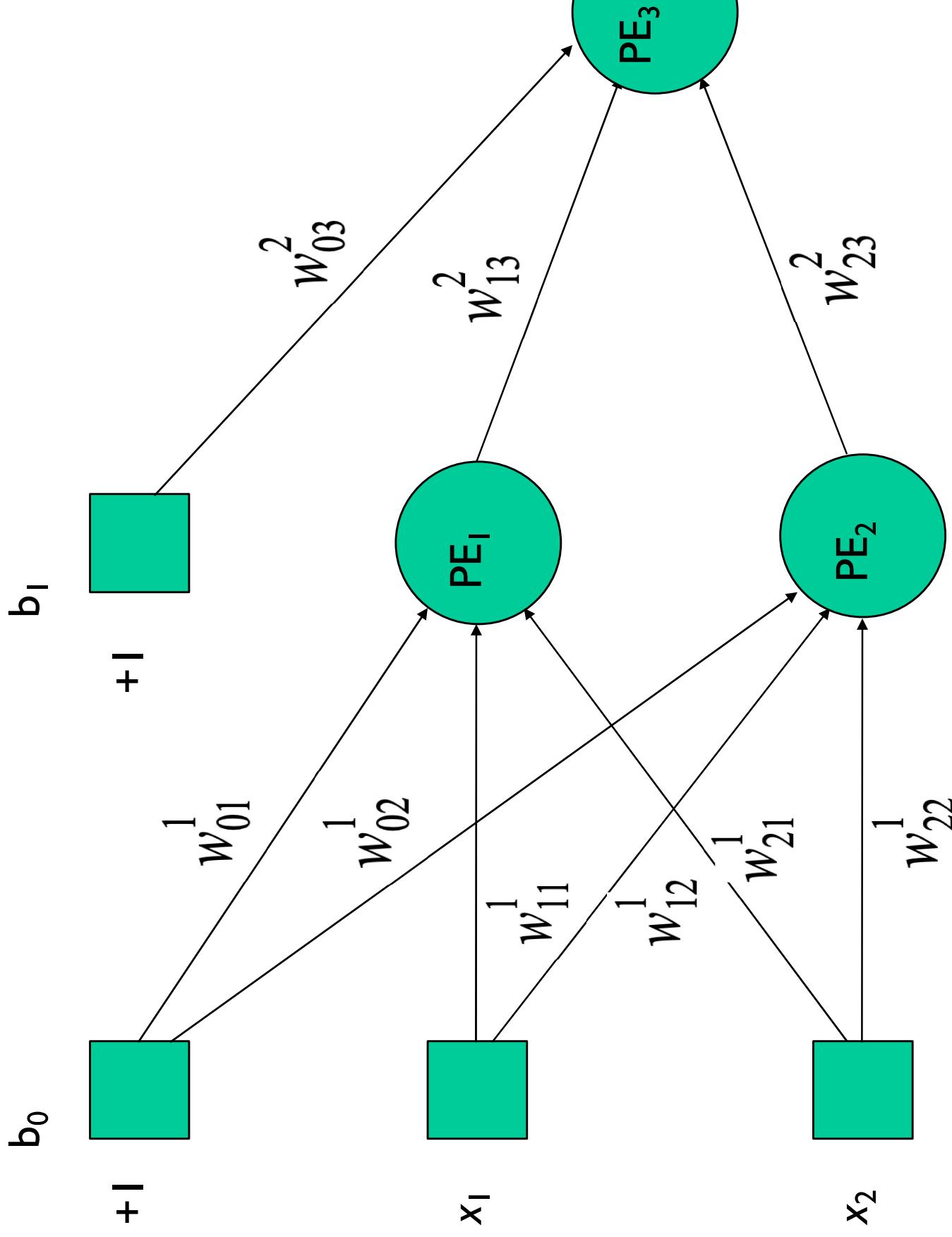


BACKPROPAGATION ALGORITHM

- Este procedimento de aprendizado é repetido várias vezes, até que para todos processadores de saída e para todos padrões de treinamento, o erro menor do que o especificado.

BACKPROPAGATION ALGORITHM

- Inicialização: pesos iniciados com valores aleatórios e pequenos ($|w_{ij}| < 0.$)
- Treinamento: loop até que o erro de cada processador de saída seja \leq tol os padrões de conjunto de treinamento (ou max Epochs)
 1. Aplica-se um padrão de entrada x_i e seu respectivo vetor de saída t_i de saída;
 2. calcula-se as saídas dos processadores, começando da primeira camada de saída;
 3. calcula-se o erro para cada processador da camada de saída
 4. atualiza os pesos de cada processador, começando pela camada de saída de entrada;
 5. volta ao passo 1.



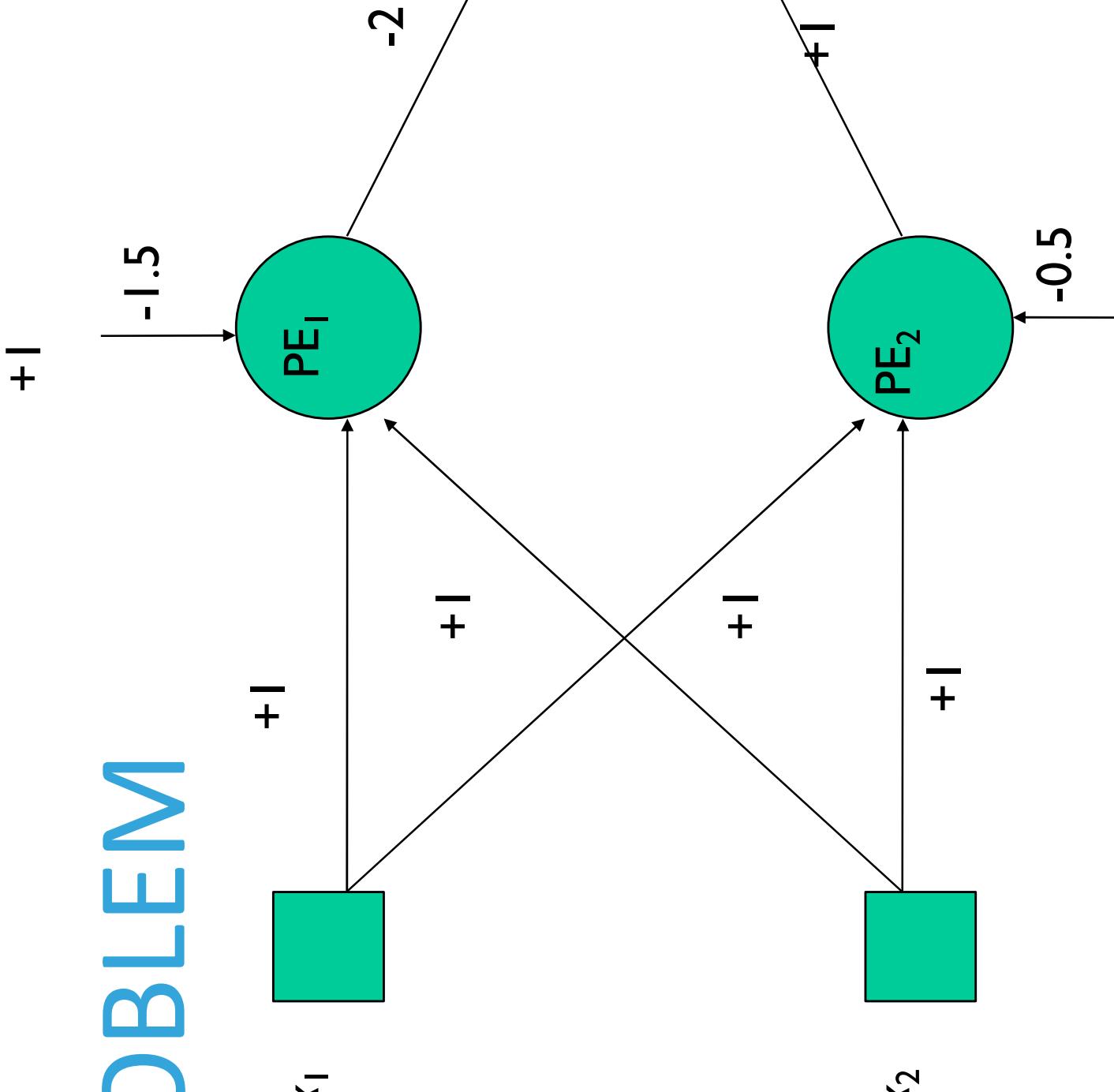
EXAMPLE

$$\begin{aligned} w_{ij} &= w_{ij} + \eta x_i \delta_j \\ \delta_j &= (t_j - x_j) F'(y_j) \\ \delta_j &= F'(y_j) \sum_k \delta_k w_{jk} \end{aligned}$$

Camada de Saída

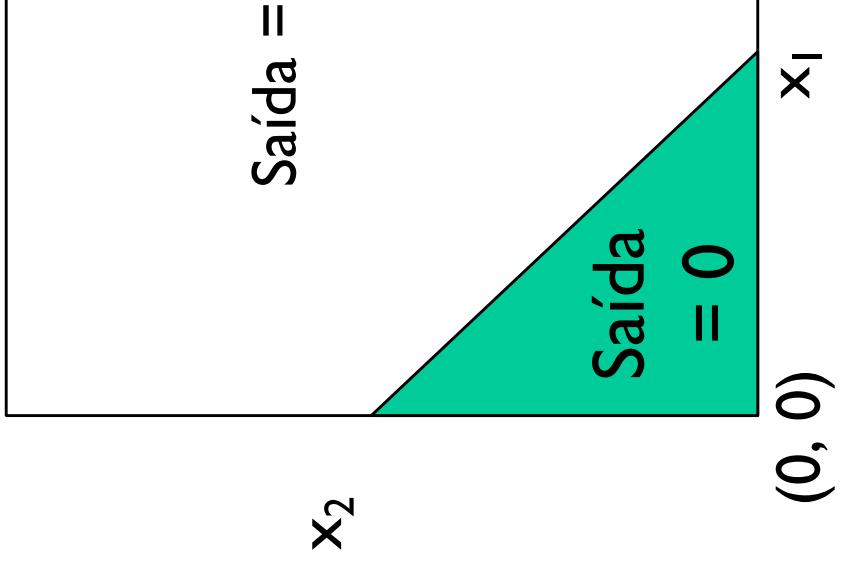
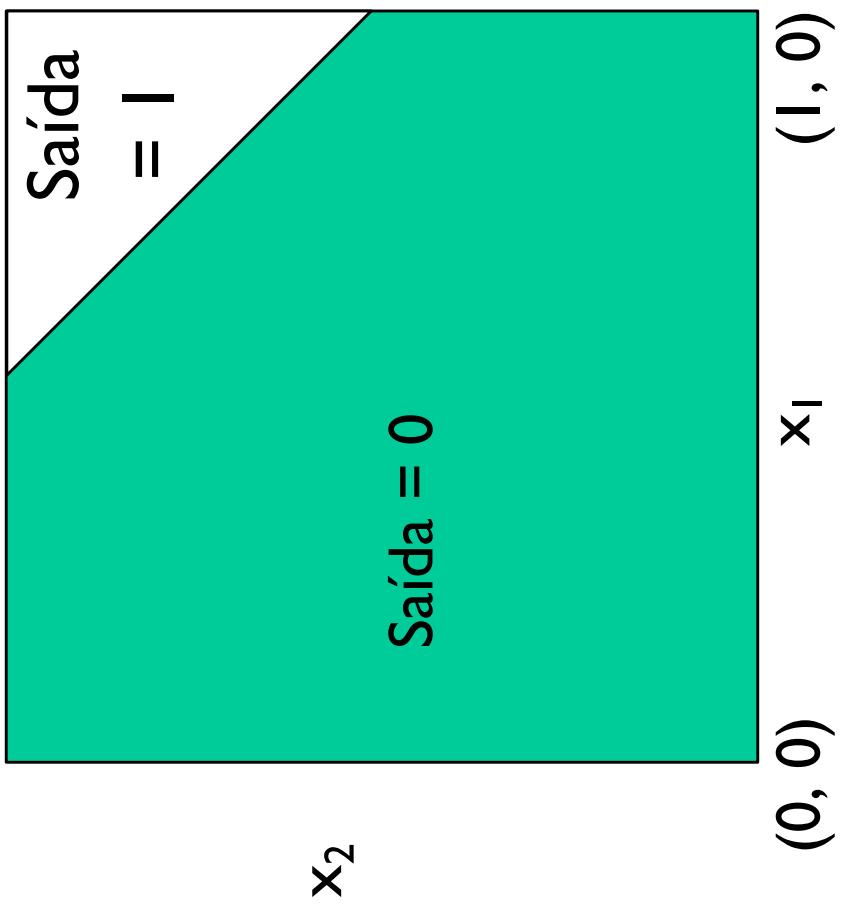
Camada Escondida

XOR PROBLEM

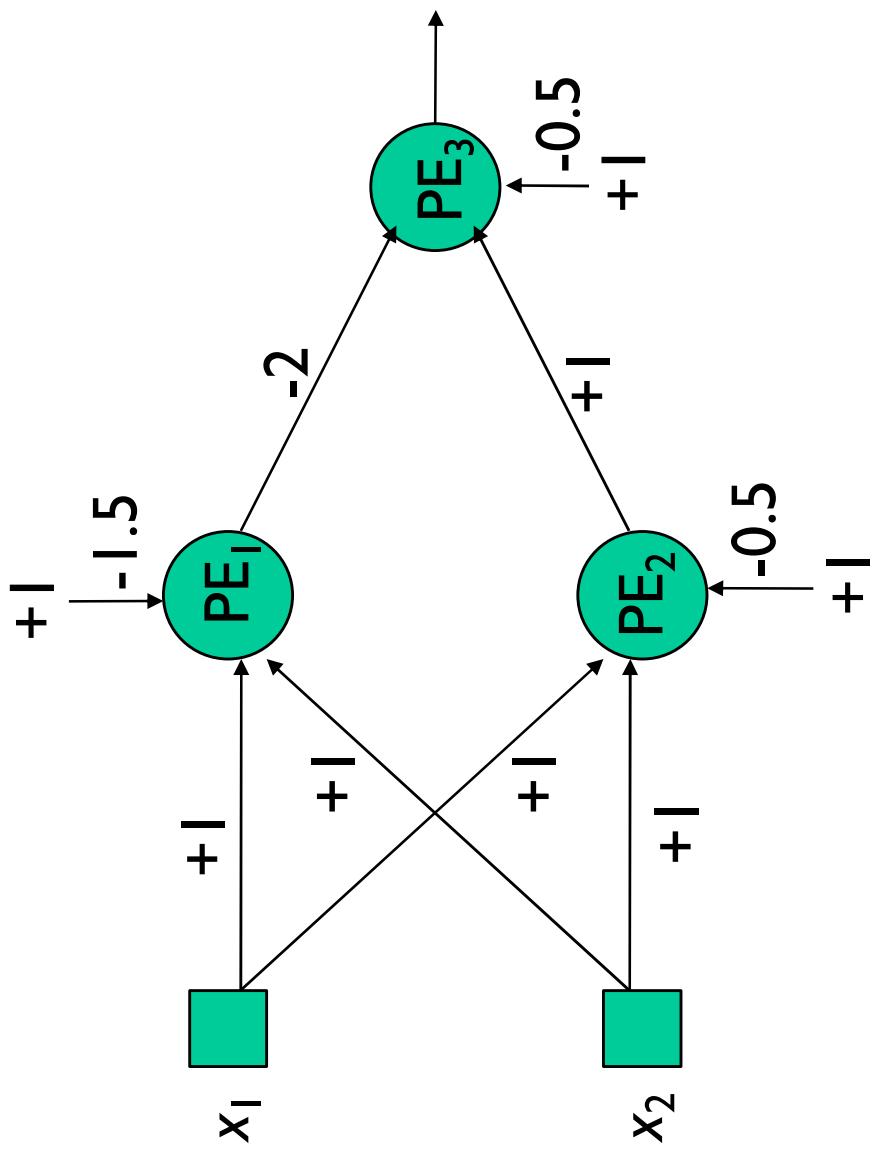


XOR PROBLEM

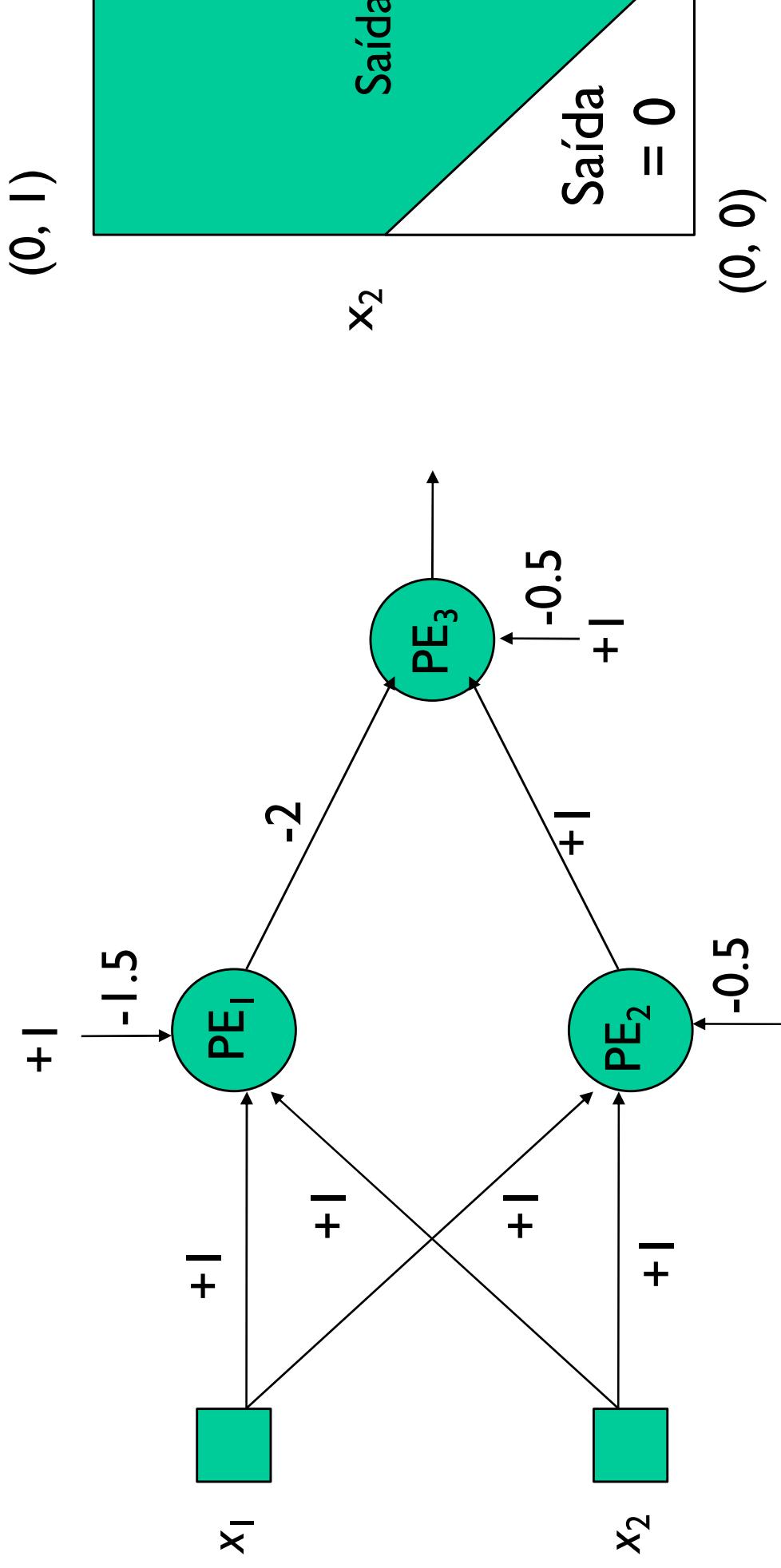
(0, 0) (0, 1)
 (1, 0)



XOR PROBLEM (O QUE ESSA REDE ESTÁ FAZENDO?)



XOR PROBLEM



FORMAS DE APRENDIZAGEM

- Aprendizado on-line (padrão)
 - ▶ Pesos atualizados após a apresentação de cada padrão da base
 - ▶ Geralmente mais rápida (**Cuidado**)
 - ▶ Pode se tornar instável
- Aprendizado em batch (lote)
 - ▶ A atualização acontece após todos os padrões terem sido apresentados
 - ▶ Apresenta uma estimativa mais precisa do vetor gradiente

FORMAS DE APRENDIZAJE

- Mini-batch: mix of both

PROJETO 1 – REDE MLP

- Selecionar dois datasets (não triviais)
- Um dataset para classificação
- Um dataset para regressão
- Separar em treino/validação/teste
- Treinar modelos MLP para os dois problemas

PROJETO 1 – REDE MLP

- Considerar:
 - Diferentes topologias ($>=5$ topologias, variar número de
 - **Usar o algoritmo original SGD (não usar algoritmos otimizados ADAM)**
 - **Avaliar o impacto do uso do Momentum**
 - **Avaliar o impacto do uso da regularização (i.e. L2)**
 - Ilustrar graficamente a evolução do treinamento (treino/validation)
 - Confeccionar um relatório (reprodutível) contendo os exper-

REFERENCES

- Livro Simon Haykin
- Paper Rummelhart et al. (1986)