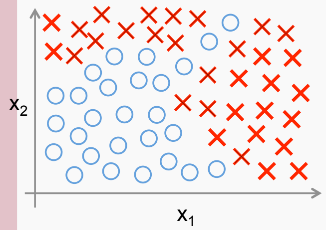
Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina

Redes neurais artificiais

Prof. Tiago A. Almeida

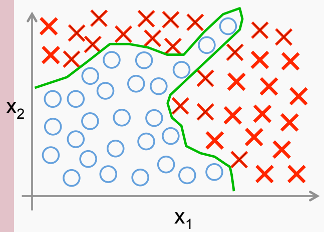
Motivação

- Aplicações reais normalmente requerem hipóteses não-lineares complexas



Motivação

- Aplicações reais normalmente requerem hipóteses não-lineares complexas



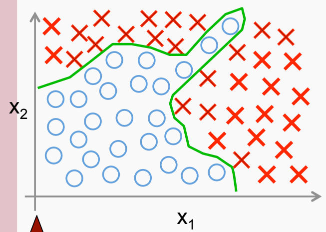
Regressão logística

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

Muitos termos polinomiais (não-lineares)!

Motivação

- Aplicações reais normalmente requerem hipóteses não-lineares complexas



2 atributos

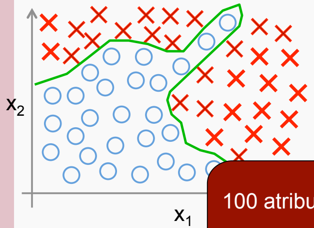
Regressão logística

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

Funciona bem com poucos atributos!

Motivação

- Aplicações reais normalmente requerem hipóteses não-lineares complexas



Regressão logística

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \ldots)$$

100 atributos e inserindo apenas termos quadráticos:
$x_1^2 + x_1x_2 + x_1x_3 + \ldots + x_1x_{100} +$
$x_2^2 + x_2x_3 + \ldots + x_2x_{100} +$
$\ldots + x_{100}^2$

~ 5000 termos ($\#\text{atributos}^2 / 2$)

Motivação

- Em aplicações reais, normalmente as amostras são formadas por muitos atributos
  - Classificação de imagens e vídeos, por exemplo

Métodos como Regressão Logística não são recomendados para ajustar hipóteses complexas não-lineares: podem consumir muito tempo e serem afetados por *overfitting*.

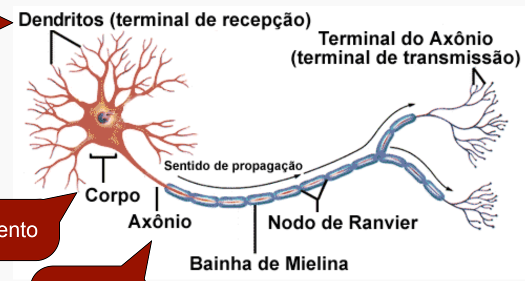
Redes neurais

- Criadas com o intuito de imitar o funcionamento do cérebro
- Tornou-se bastante popular entre 1980-1990
- Popularidade reduziu no final da década de 1990 (alto custo computacional)
- Com o aumento do poder computacional, tornou-se popular novamente e atualmente é considerada o estado-da-arte em muitas aplicações

Redes neurais - Representação

- Criadas para simular **neurônios**

entrada

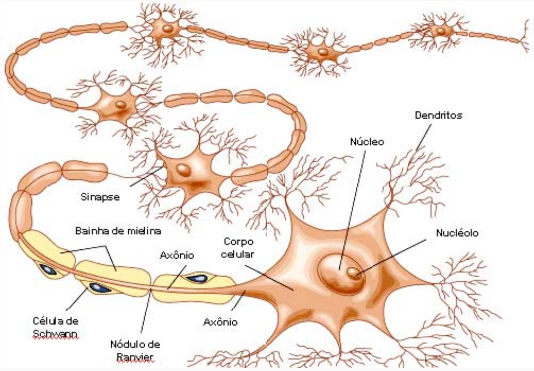


processamento

saída

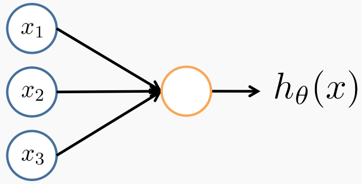
Redes neurais - Representação

- Rede neural biológica



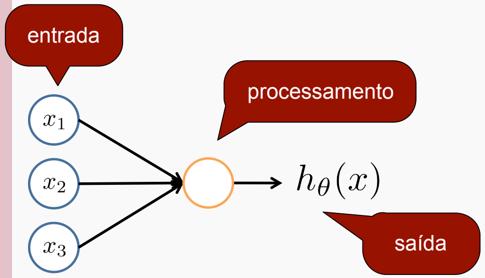
Redes neurais - Representação

- **Modelo neural: unidade lógica**



Redes neurais - Representação

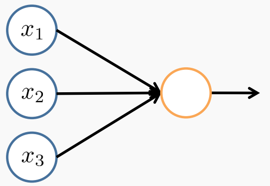
- **Modelo neural: unidade lógica**



Redes neurais - Representação

- **Modelo neural: unidade lógica**

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

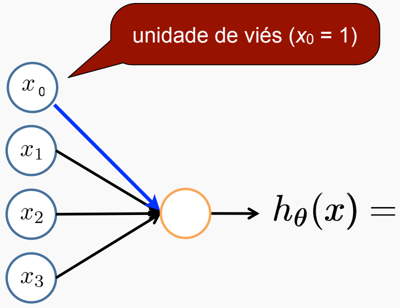
pesos

Função de ativação sigmoidal (logística)

Redes neurais - Representação

- **Modelo neural: unidade lógica**

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

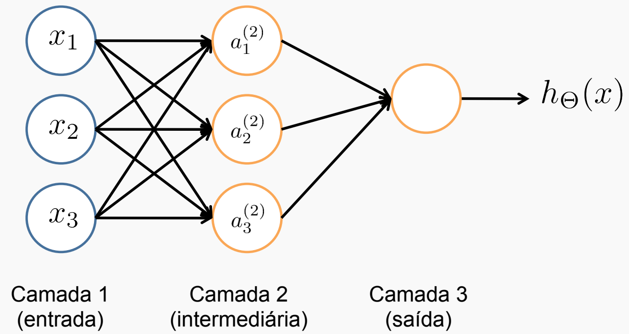


$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Função de ativação sigmoidal (logística)

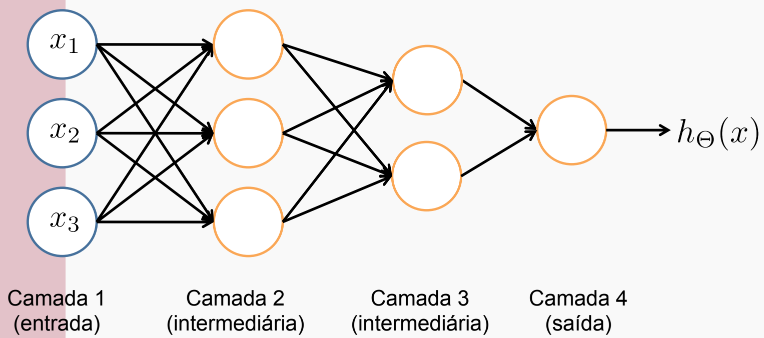
Redes neurais - Arquitetura (1)

- **Rede neural artificial**



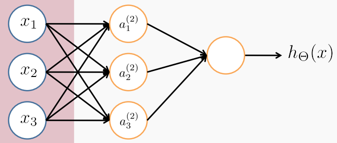
Redes neurais - Arquitetura (2)

- **Rede neural artificial**



Redes neurais - Representação

- **Rede neural artificial**

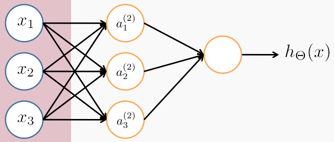


$a_i^{(j)}$ = valor de ativação da unidade *i* na camada *j*

$\Theta^{(j)}$ = matriz de pesos de controle da função de mapeamento da camada *j* para a camada *j + 1*

Redes neurais - Representação

- **Rede neural artificial**



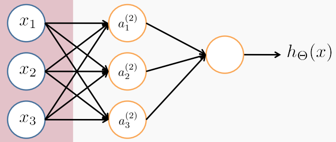
$a_i^{(j)}$ = valor de ativação da unidade *i* na camada *j*

$\Theta^{(j)}$ = matriz de pesos de controle da função de mapeamento da camada *j* para a camada *j + 1*

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Redes neurais - Representação

- **Rede neural artificial**



$a_i^{(j)}$ = valor de ativação da unidade *i* na camada *j*

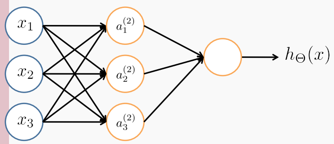
$\Theta^{(j)}$ = matriz de pesos de controle da função de mapeamento da camada *j* para a camada *j + 1*

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Se a rede contém t_j unidades na camada *j*, t_{j+1} unidades na camada *j* + 1, então $\Theta^{(j)}$ terá dimensão t_{j+1} x (t_j + 1)

Redes neurais - Representação

- **Rede neural artificial: terminologia**



$$a_1^{(2)} = g(\boxed{\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3})$$
$$a_2^{(2)} = g(\boxed{\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3})$$
$$a_3^{(2)} = g(\boxed{\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3})$$
$$h_\Theta(x) = a_1^{(3)} = g(\boxed{\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}})$$

$z_1^{(2)}$ — $a_1^{(2)} = g(z_1^{(2)})$

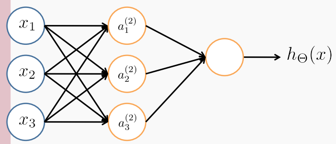
$z_2^{(2)}$ — $a_2^{(2)} = g(z_2^{(2)})$

$z_2^{(2)}$ — $a_3^{(2)} = g(z_3^{(2)})$

$z^{(3)}$ — $a^{(3)} = g(z^{(3)})$

Redes neural: Forward Propagation

- **Rede neural artificial: cálculo**

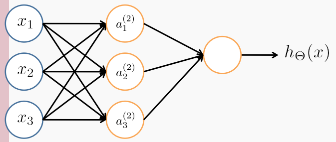


$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

Redes neural: Forward Propagation

- **Rede neural artificial: cálculo**



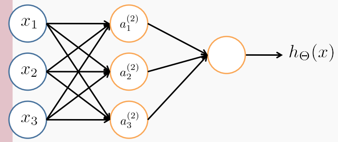
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

Redes neural: Forward Propagation

- **Rede neural artificial: cálculo**



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

adicionar $a_0^{(2)} = 1$

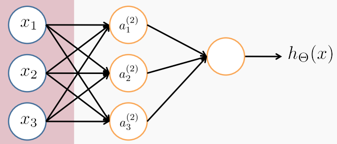
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina

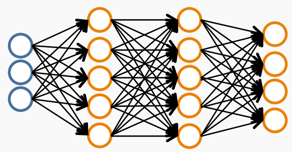
Redes neurais artificiais: aprendizado

Prof. Tiago A. Almeida

Redes neurais - Classificação



Classificação binária (y = 0 ou 1)

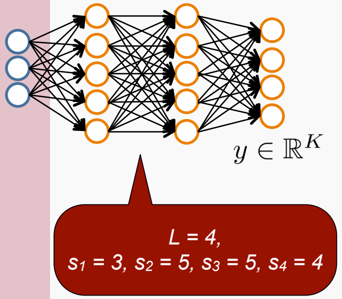


Multi-classes (K classes)

$y \in \mathbb{R}^K$ Ex. $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Classe A, Classe B, Classe C, Classe D

Redes neurais - Notação



$L =$ quantidade de camadas

$s_l =$ n. de unidades (excluindo bias) na camada *l*

Redes neurais - Função Custo

- **Regressão Logística**

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- **Rede Neural**

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i \text{ -ésima saída}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Redes neurais - Backpropagation

- **Rede Neural**

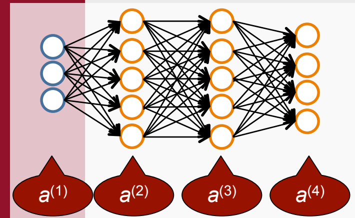
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$$\min_\Theta J(\Theta)$$

Computar:

$$J(\Theta)$$
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

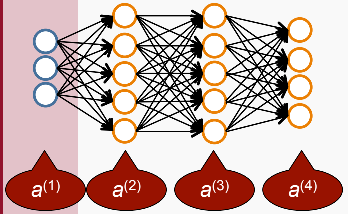
Redes neurais - Gradiente



Seja um único exemplo (*x*, *y*)

Tiago A. Almeida

Redes neurais - Gradiente



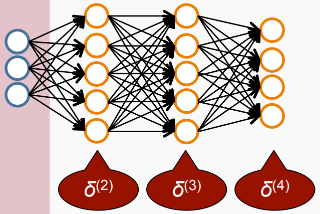
Seja um único exemplo (*x*, *y*)

Forward propagation:

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad \text{(adicionar } a_0^{(2)}\text{)} \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \quad \text{(adicionar } a_0^{(3)}\text{)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

Tiago A. Almeida

Redes neurais - Gradiente



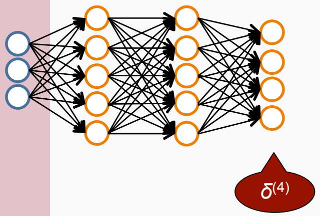
<u>Ideia</u>:

Calcular $\delta_j^{(l)}$ **= "erro" produzido**

por cada nó *j* da camada *l*.

Tiago A. Almeida

Redes neurais - Gradiente



Backpropagation:

Para cada unidade de saída (L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

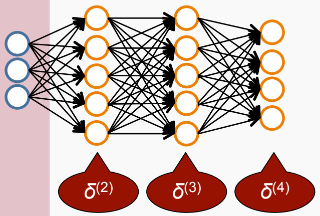
<u>Ideia</u>:

Calcular $\delta_j^{(l)}$ **= "erro" produzido**

por cada nó *j* da camada *l*.

Tiago A. Almeida

Redes neurais - Gradiente



Backpropagation:

Para cada unidade de saída (L = 4)

$$
\begin{aligned}
\delta_j^{(4)} &= a_j^{(4)} - y_j \\
\delta^{(3)} &= (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \\
\delta^{(2)} &= (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})
\end{aligned}
$$

derivada parcial de
$g(z^{(l)})$ => $a^{(l)} .* (1 - a^{(l)})$

<u>Ideia</u>:

Calcular $\delta_j^{(l)}$ **= "erro" produzido**

por cada nó *j* da camada *l*.

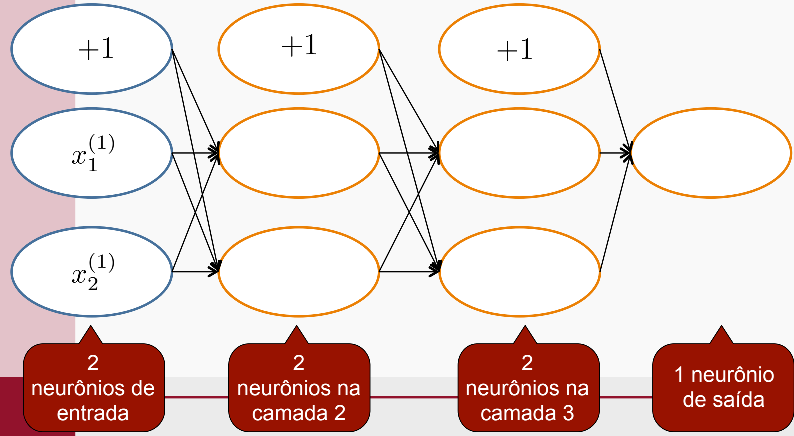
Redes neurais - Backpropagation

- Base de treinamento: $\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$
- Inicializar $\triangle_{ij}^{(l)}=0$ (para todo l,i,j)
- Para $i=1\ :\ m$
  - $a^{(1)}=x^{(i)}$
  - Aplicar *Forward propagation* para calcular $a^{(l)}$ para $l=2,3,\ldots,L$
  - Calcular $\delta^{(L)}=a^{(L)}-y^{(i)}$
  - Calcular $\delta^{(L-1)},\delta^{(L-2)},\ldots,\delta^{(2)}$
  - Acumular as derivadas parciais de cada exemplo: $\triangle_{ij}^{(l)}:=\triangle_{ij}^{(l)}+a_j^{(l)}\delta_i^{(l+1)}$
- Calcular a derivada da função custo:

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)=D_{ij}^{(l)} \quad \longrightarrow \quad \begin{array}{ll} D_{ij}^{(l)}:=\frac{1}{m}\triangle_{ij}^{(l)}+\lambda\Theta_{ij}^{(l)} & \text{se } j\neq 0 \\ D_{ij}^{(l)}:=\frac{1}{m}\triangle_{ij}^{(l)} & \text{se} \quad \text{if } j=0 \end{array}$$

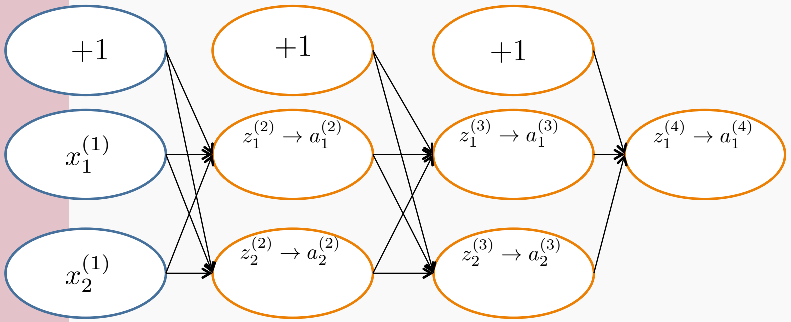
Rede neural: Prática

- **Forward propagation**



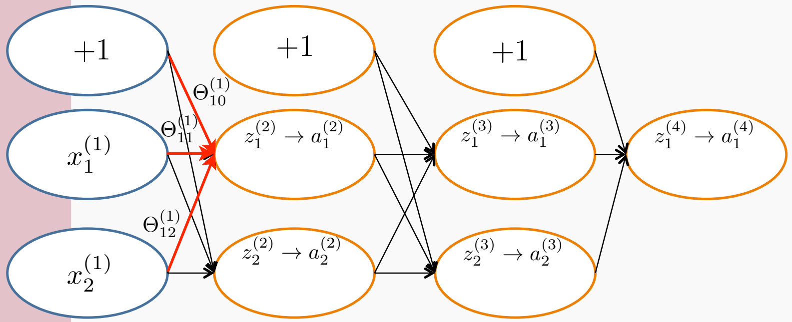
Rede neural: Prática

- **Forward propagation**



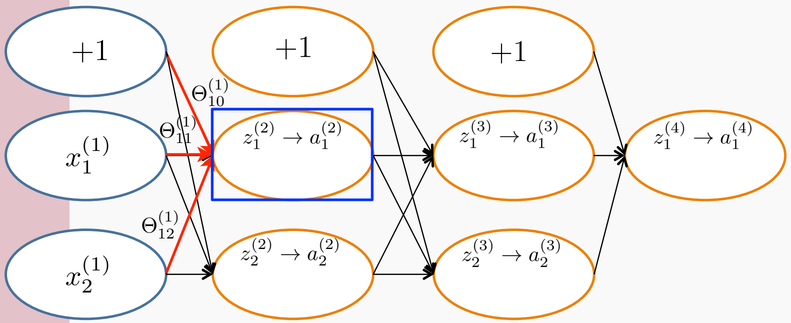
Rede neural: Prática

- **Forward propagation**



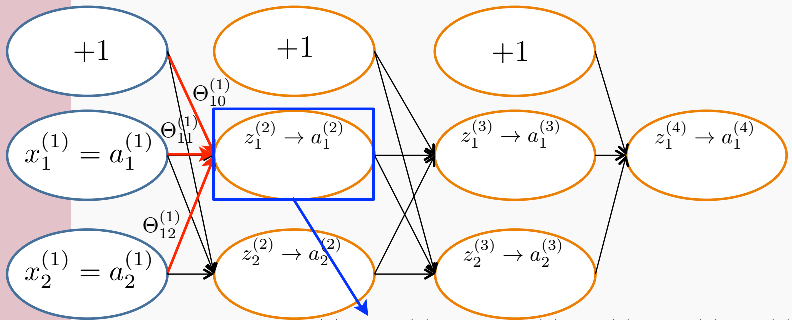
Rede neural: Prática

- **Forward propagation**



Rede neural: Prática

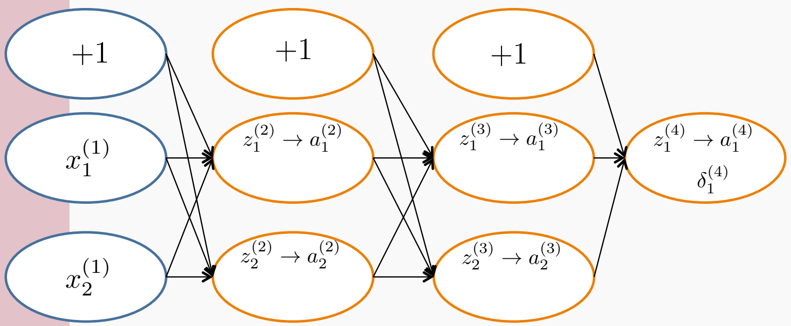
- **Forward propagation**



$$z_1^{(2)} = \Theta_{10}^{(1)} * 1 + \Theta_{11}^{(1)} * a_1^{(1)} + \Theta_{12}^{(1)} * a_2^{(1)}$$

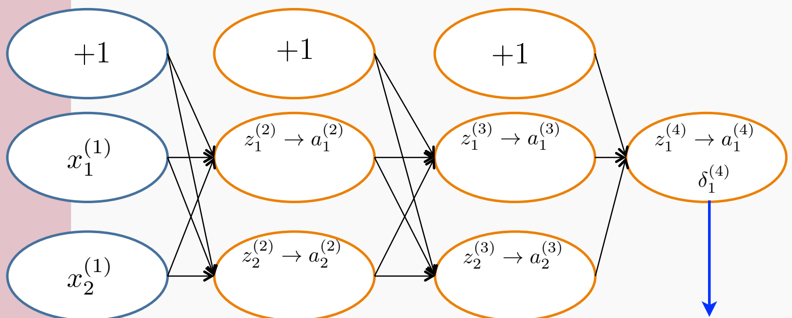
Rede neural: Prática

- **Backpropagation**



Rede neural: Prática

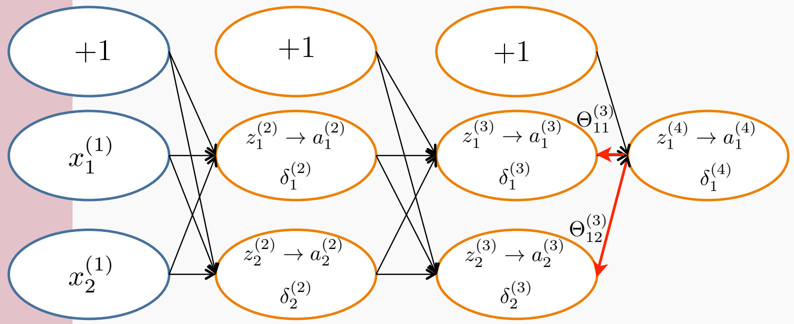
- **Backpropagation**



$$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$$

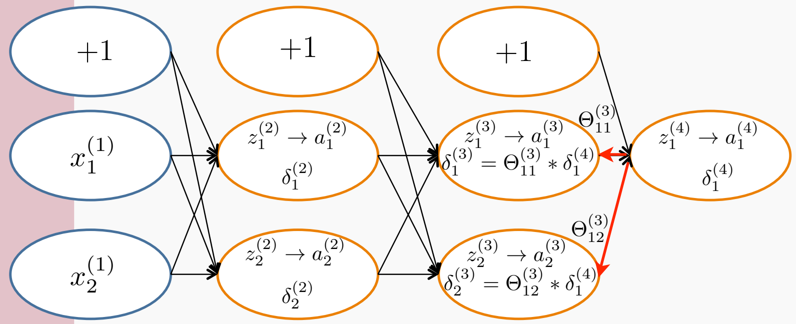
Rede neural: Prática

- **Backpropagation**



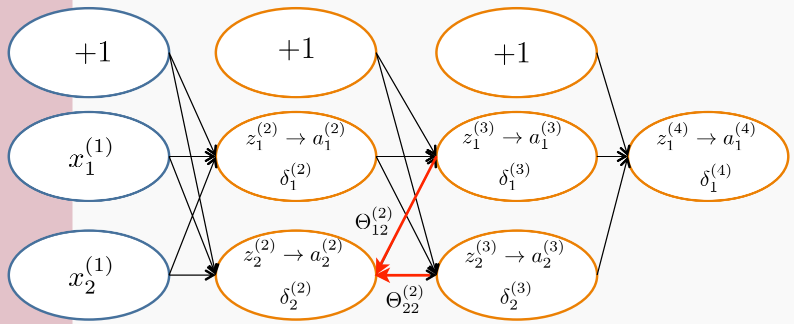
Rede neural: Prática

- **Backpropagation**



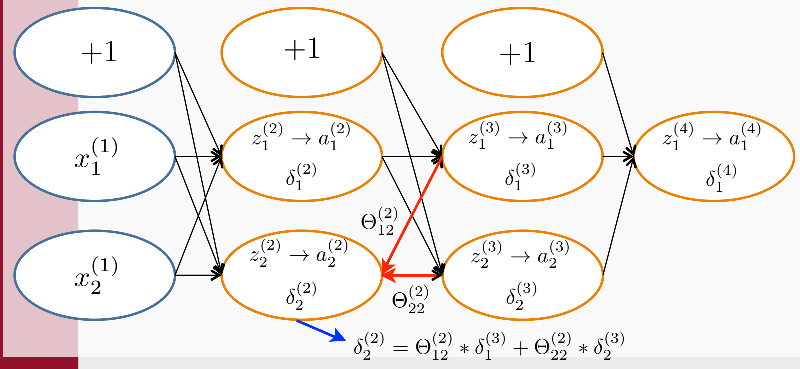
Rede neural: Prática

- **Backpropagation**



Rede neural: Prática

- **Backpropagation**



Redes neurais: resumo e conselhos

- **1. Definir a arquitetura da rede**
  - **No. de neurônios na entrada:** quantidade de atributos por amostra
  - **No. de neurônios na saída:** quantidade de classes
  - **No. de camadas intermediárias:** normalmente 1 camada intermediária
  - **No. de neurônios nas camadas intermediárias:** quantidade igual em todas as camadas intermediárias (no. de neurônios maior que camadas de E/S). Obs: quanto mais neurônios e/ou camadas, maior será o esforço computacional.

Redes neurais: resumo e conselhos

- **2. Treinamento da rede**
  - Inicializar pesos com valores aleatórios próximos de zero
  - Implementar *forward propagation* para obter $h_\Theta(x^{(i)})$ para cada $x^{(i)}$
  - Implementar função custo $J(\Theta)$
  - Implementar *backpropagation* para obter as derivadas parciais $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

Redes neurais: resumo e conselhos

- **2. Treinamento da rede**
  - Inicializar pesos com valores aleatórios próximos de zero
  - Implementar *forward propagation* para obter $h_\Theta(x^{(i)})$ para cada $x^{(i)}$
  - Implementar função custo $J(\Theta)$
  - Implementar *backpropagation* para obter as derivadas parciais $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
    - Para cada amostra da base *i* = 1 : *m*
      - Executar *forward propagation* e *backpropagation* usando $(x^{(i)}, y^{(i)})$ para obter os valores de ativação $a^{(l)}$ e os erros $\delta^{(l)}$ para $l = 2, \ldots, L$
      - Acumular os erros: $\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
    - Computar $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

Redes neurais: resumo e conselhos

- **2. Treinamento da rede**
  - Inicializar pesos com valores aleatórios próximos de zero
  - Implementar *forward propagation* para obter $h_\Theta(x^{(i)})$ para cada $x^{(i)}$
  - Implementar função custo $J(\Theta)$
  - Implementar *backpropagation* para obter as derivadas parciais $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
    - Para cada amostra da base *i* = 1 : *m*
      - Executar *forward propagation* e *backpropagation* usando $(x^{(i)}, y^{(i)})$ para obter os valores de ativação $a^{(l)}$ e os erros $\delta^{(l)}$ para $l = 2, \ldots, L$
      - Acumular os erros: $\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
    - Computar $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
  - Empregar método de otimização para setar parâmetros Θ que minimizem $J(\Theta)$

Objetos de pesquisa

- *Deep Learning*
- *Extreme Learning Machines*