Aula 8 - Redes Neurais II (*Backpropagation*)

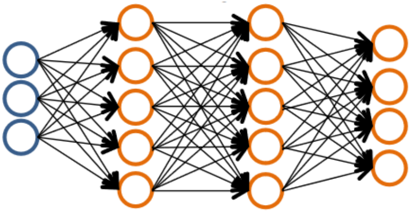
João Florindo

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas - Brasil
florindo@unicamp.br

Outline

Notação

- Treinamento: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
- L: Número total de camadas da rede
- n_l: Número de unidades (neurônios) (excluindo o *bias*) na camada l
- K: Número de classes (unidades de saída)



Acima temos $L = 4$, $n_1 = 3$, $n_2 = 5$, $n_4 = n_L = 4$, $K = 4$.

Lembrando também do exemplo visto:

$$\underset{\text{pedestre}}{y^{(i)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}} \quad \underset{\text{carro}}{y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}} \quad \underset{\text{moto}}{y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}} \quad \underset{\text{caminhão}}{y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

Função de Custo

- Cada unidade de saída faz uma regressão logística:

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

- Função de custo da rede é a soma do custo de todas as K saídas:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)})_k) + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)})_k))\right],$$

  em que $h_\Theta(x)_k$ é a k-ésima saída da função de hipótese e y_k é o k-ésimo componente do vetor de saída y.

Função de Custo

- Cada unidade de saída faz uma regressão logística:

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

- Função de custo da rede é a soma do custo de todas as K saídas:

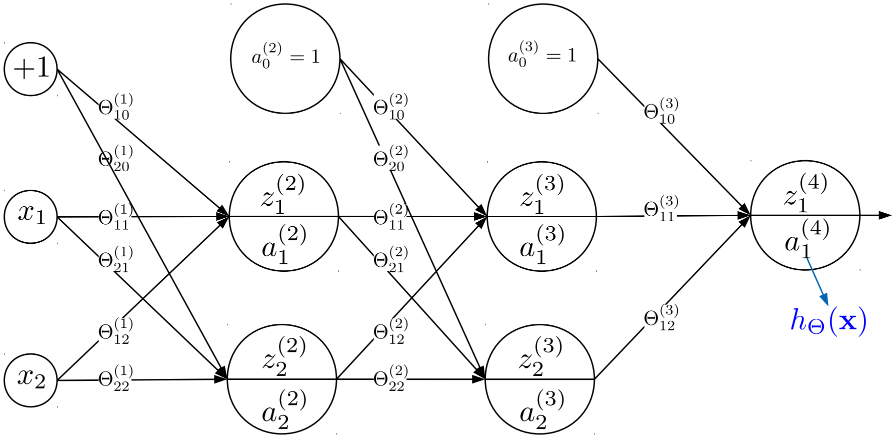
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)})_k) + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)})_k))\right],$$

em que $h_\Theta(x)_k$ é a k-ésima saída da função de hipótese e y_k é o k-ésimo componente do vetor de saída y.

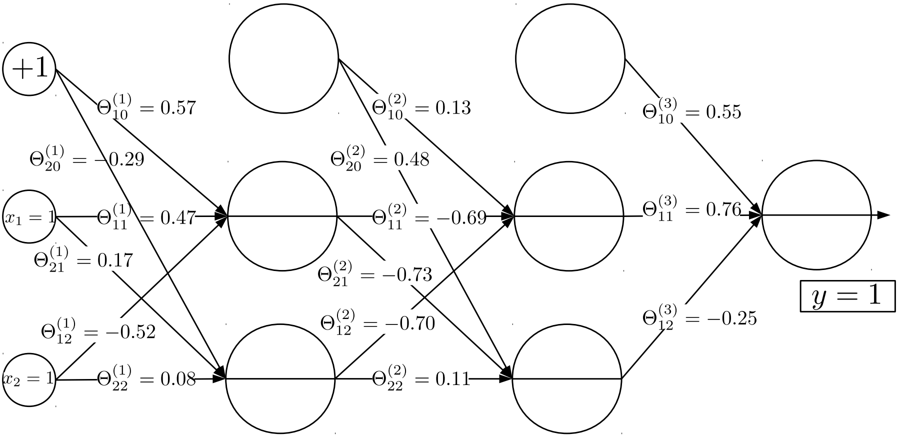
Observação: Custo $\times$ Perda

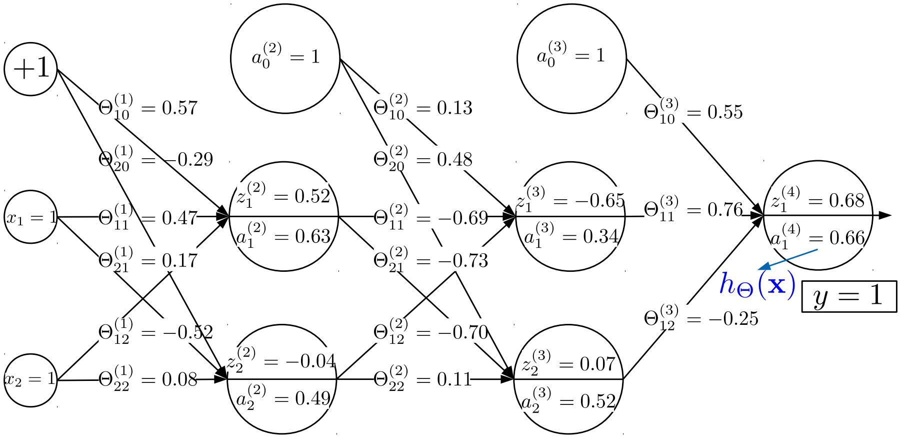
- Em redes neurais é comum que apareça o termo *loss function* (função de perda).
- A *loss function* é a função de custo para um único exemplo de treinamento.

Outline



$+1$
x_1
x_2
$a_0^{(2)} = 1$
$a_0^{(3)} = 1$
$z_1^{(2)}$
$a_1^{(2)}$
$z_2^{(2)}$
$a_2^{(2)}$
$z_1^{(3)}$
$a_1^{(3)}$
$z_2^{(3)}$
$a_2^{(3)}$
$z_1^{(4)}$
$a_1^{(4)}$
$h_\Theta(\mathbf{x})$
$\Theta_{10}^{(1)}$
$\Theta_{20}^{(1)}$
$\Theta_{11}^{(1)}$
$\Theta_{21}^{(1)}$
$\Theta_{12}^{(1)}$
$\Theta_{22}^{(1)}$
$\Theta_{10}^{(2)}$
$\Theta_{20}^{(2)}$
$\Theta_{11}^{(2)}$
$\Theta_{21}^{(2)}$
$\Theta_{12}^{(2)}$
$\Theta_{22}^{(2)}$
$\Theta_{10}^{(3)}$
$\Theta_{11}^{(3)}$
$\Theta_{12}^{(3)}$


+1
$x_1 = 1$
$x_2 = 1$
$\Theta^{(1)}_{10} = 0.57$
$\Theta^{(1)}_{20} = -0.29$
$\Theta^{(1)}_{11} = 0.47$
$\Theta^{(1)}_{21} = 0.17$
$\Theta^{(1)}_{12} = -0.52$
$\Theta^{(1)}_{22} = 0.08$
$\Theta^{(2)}_{10} = 0.13$
$\Theta^{(2)}_{20} = 0.48$
$\Theta^{(2)}_{11} = -0.69$
$\Theta^{(2)}_{21} = -0.73$
$\Theta^{(2)}_{12} = -0.70$
$\Theta^{(2)}_{22} = 0.11$
$\Theta^{(3)}_{10} = 0.55$
$\Theta^{(3)}_{11} = 0.76$
$\Theta^{(3)}_{12} = -0.25$
$y = 1$



+1
$x_1 = 1$
$x_2 = 1$
$\Theta^{(1)}_{10} = 0.57$
$\Theta^{(1)}_{20} = -0.29$
$\Theta^{(1)}_{11} = 0.47$
$\Theta^{(1)}_{21} = 0.17$
$\Theta^{(1)}_{12} = -0.52$
$\Theta^{(1)}_{22} = 0.08$
$a^{(2)}_0 = 1$
$z^{(2)}_1 = 0.52$
$a^{(2)}_1 = 0.63$
$z^{(2)}_2 = -0.04$
$a^{(2)}_2 = 0.49$
$\Theta^{(2)}_{10} = 0.13$
$\Theta^{(2)}_{20} = 0.48$
$\Theta^{(2)}_{11} = -0.69$
$\Theta^{(2)}_{21} = -0.73$
$\Theta^{(2)}_{12} = -0.70$
$\Theta^{(2)}_{22} = 0.11$
$a^{(3)}_0 = 1$
$z^{(3)}_1 = -0.65$
$a^{(3)}_1 = 0.34$
$z^{(3)}_2 = 0.07$
$a^{(3)}_2 = 0.52$
$\Theta^{(3)}_{10} = 0.55$
$\Theta^{(3)}_{11} = 0.76$
$\Theta^{(3)}_{12} = -0.25$
$z^{(4)}_1 = 0.68$
$a^{(4)}_1 = 0.66$
$h_\Theta(\mathbf{x})$
$y = 1$

- Definimos $\delta_j^{(l)}$: "erro" do nó j na camada l.
- O erro é calculado de trás para frente (daí o nome "*backpropagation*").
- Na unidade de saída:

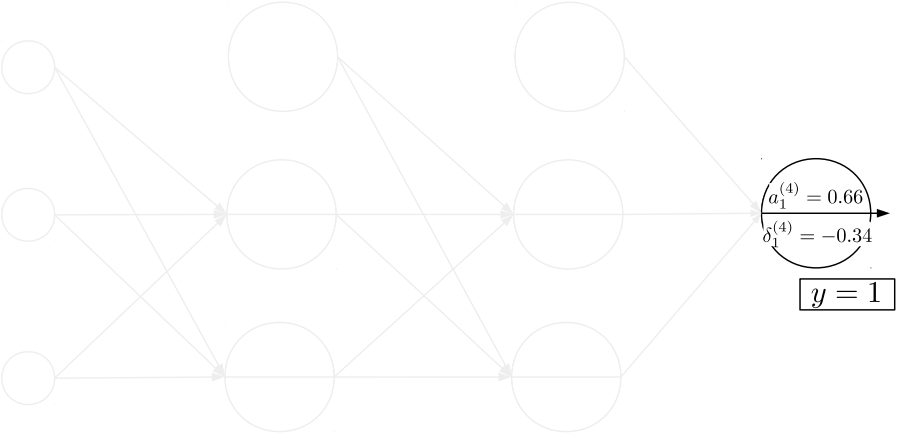
$$\delta_i^{(L)} = a_i^{(L)} - y.$$

- Definimos $\delta_j^{(l)}$: "erro" do nó j na camada l.
- O erro é calculado de trás para frente (daí o nome "*backpropagation*").
- Na unidade de saída:

$$\delta_i^{(L)} = a_i^{(L)} - y.$$

- Definimos $\delta_j^{(l)}$: “erro” do nó j na camada l.
- O erro é calculado de trás para frente (daí o nome “*backpropagation*”).
- Na unidade de saída:

$$\delta_i^{(L)} = a_i^{(L)} - y.$$



$$\delta_1^{(4)} = a_1^{(4)} - y = 0.66 - 1 = -0.34.$$

- Na Camada 3:
$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} g'(z_i^{(3)}),$$
em que $g'(z)$ é a derivada da sigmoide $g(z)$ ponto-a-ponto.

- Mas lembre-se que $g'(z_i^{(3)}) = g(z_i^{(3)})(1 - g(z_i^{(3)})) = a_i^{(3)}(1 - a_i^{(3)})$. Portanto:
$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} a_i^{(3)}(1 - a_i^{(3)}).$$

- NOTA: Não calculamos $\delta_0^{(l)}$ para nenhuma camada l.

- Na Camada 3:

$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} g'(z_i^{(3)}),$$

em que $g'(z)$ é a derivada da sigmoide $g(z)$ ponto-a-ponto.

- Mas lembre-se que $g'(z_i^{(3)}) = g(z_i^{(3)})(1 - g(z_i^{(3)})) = a_i^{(3)}(1 - a_i^{(3)})$. Portanto:

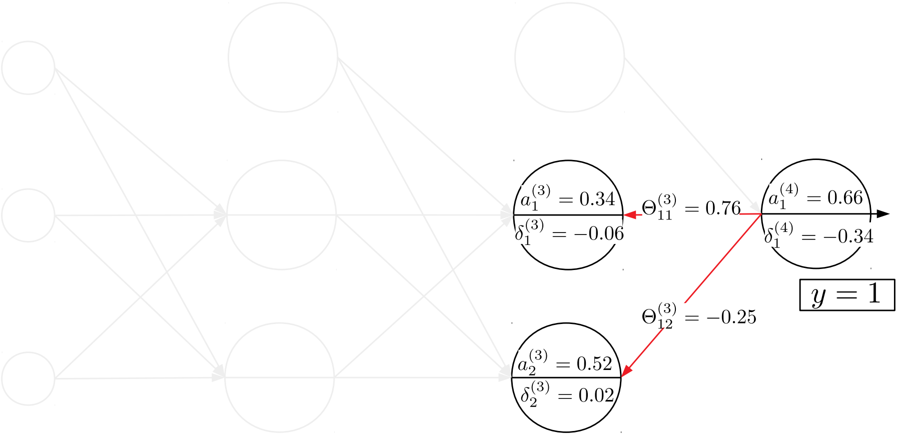
$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} a_i^{(3)}(1 - a_i^{(3)}).$$

- NOTA: Não calculamos $\delta_0^{(l)}$ para nenhuma camada l.

- Na Camada 3:
$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} g'(z_i^{(3)}),$$
em que $g'(z)$ é a derivada da sigmoide $g(z)$ ponto-a-ponto.

- Mas lembre-se que $g'(z_i^{(3)}) = g(z_i^{(3)})(1 - g(z_i^{(3)})) = a_i^{(3)}(1 - a_i^{(3)})$.
Portanto:
$$\delta_i^{(3)} = \Theta_{1i}^{(3)} \delta_1^{(4)} a_i^{(3)} (1 - a_i^{(3)}).$$

- NOTA: Não calculamos $\delta_0^{(l)}$ para nenhuma camada l.

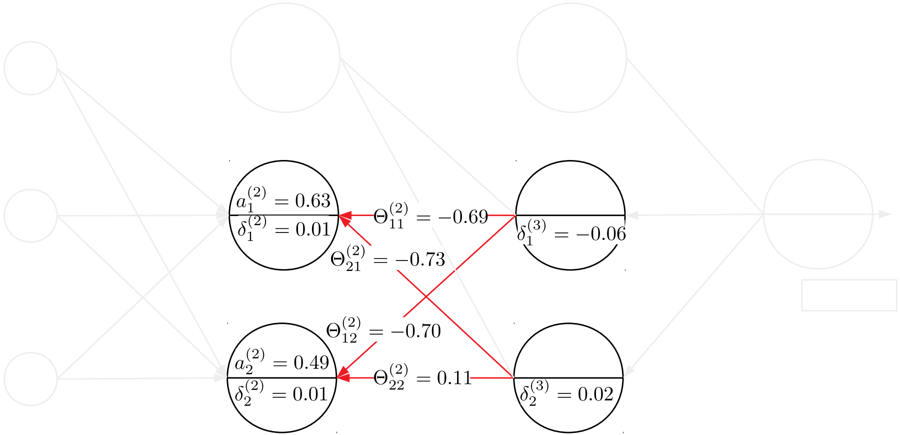


$$\delta_1^{(3)} = \Theta_{11}^{(3)}\delta_1^{(4)}a_1^{(3)}(1 - a_1^{(3)}) = 0.76 \cdot (-0.34) \cdot 0.34 \cdot (1 - 0.34) = -0.06$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)}\delta_1^{(4)}a_2^{(3)}(1 - a_2^{(3)}) = (-0.25) \cdot (-0.34) \cdot 0.52 \cdot (1 - 0.52) = 0.02$$

- Na Camada 2, incluímos os δ's que se conectam na camada seguinte:

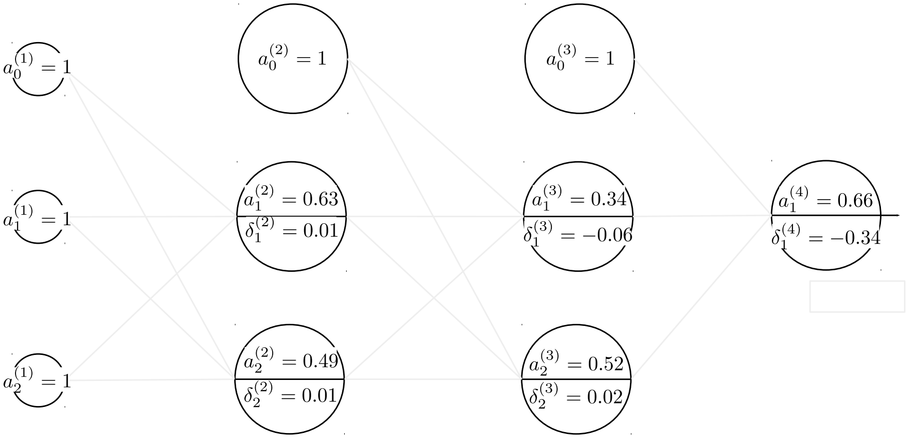
$$\delta_i^{(2)} = \left(\sum_{j=1}^{n_3} \Theta_{ji}^{(2)} \delta_j^{(3)} \right) a_i^{(2)} (1 - a_i^{(2)}).$$



$$\delta_1^{(2)} = (\Theta_{11}^{(2)}\delta_1^{(3)} + \Theta_{21}^{(2)}\delta_2^{(3)})a_1^{(2)}(1 - a_1^{(2)}) = ((-0.69)\cdot(-0.06) + (-0.73)\cdot 0.002)\cdot 0.63\cdot(1 - 0.63) = 0.01$$

$$\delta_2^{(2)} = (\Theta_{12}^{(2)}\delta_1^{(3)} + \Theta_{22}^{(2)}\delta_2^{(3)})a_2^{(2)}(1 - a_2^{(2)}) = ((-0.70)\cdot(-0.06) + (0.11)\cdot 0.002)\cdot 0.49\cdot(1 - 0.49) = 0.01$$

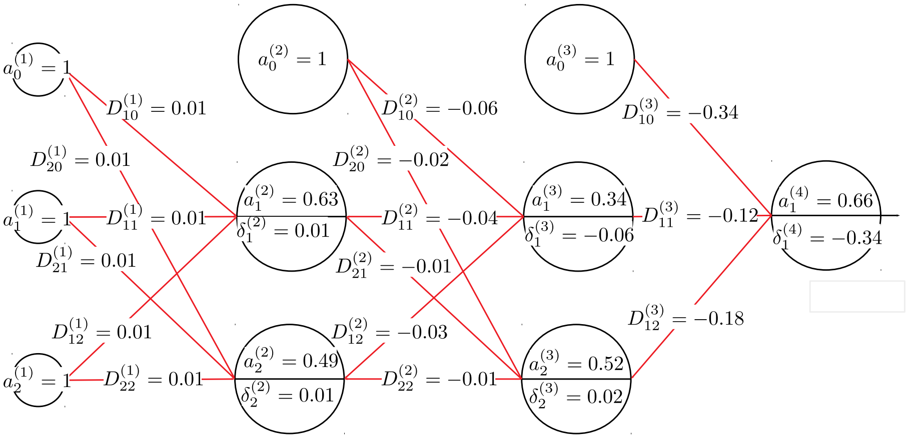
- Juntando as partes.



$a_0^{(1)} = 1$
$a_1^{(1)} = 1$
$a_2^{(1)} = 1$
$a_0^{(2)} = 1$
$a_1^{(2)} = 0.63$
$\delta_1^{(2)} = 0.01$
$a_2^{(2)} = 0.49$
$\delta_2^{(2)} = 0.01$
$a_0^{(3)} = 1$
$a_1^{(3)} = 0.34$
$\delta_1^{(3)} = -0.06$
$a_2^{(3)} = 0.52$
$\delta_2^{(3)} = 0.02$
$a_1^{(4)} = 0.66$
$\delta_1^{(4)} = -0.34$

- Derivadas:

$$D_{ij} = \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.$$



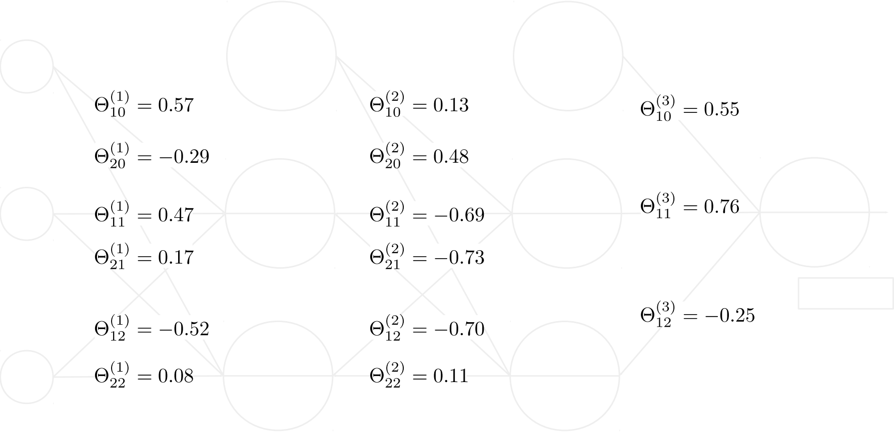
$$D_{10}^{(1)} = a_0^{(1)}\delta_1^{(2)} = 1 \cdot 0.01 = 0.01$$

$$D_{11}^{(2)} = a_1^{(2)}\delta_1^{(3)} = 0.63 \cdot (-0.06) = -0.04$$

$$D_{12}^{(3)} = a_2^{(3)}\delta_1^{(4)} = 0.52 \cdot (-0.34) = -0.18$$

$$\cdots$$

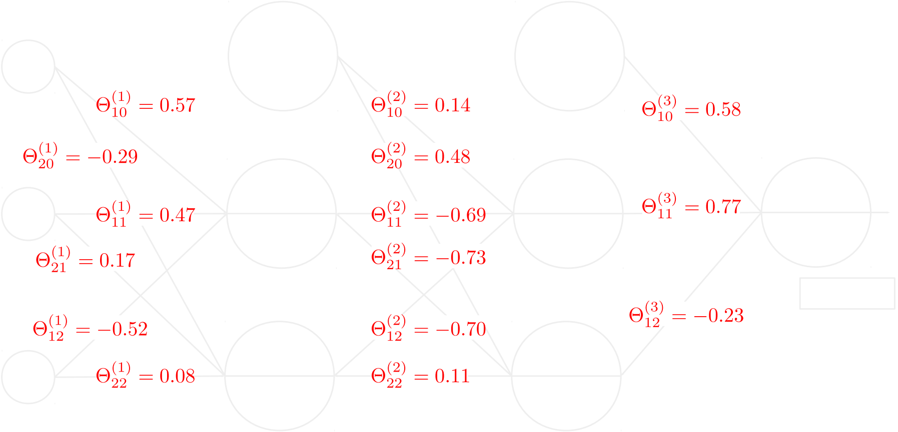
- Lembrando dos pesos originais.


$\Theta^{(1)}_{10} = 0.57$
$\Theta^{(1)}_{20} = -0.29$
$\Theta^{(1)}_{11} = 0.47$
$\Theta^{(1)}_{21} = 0.17$
$\Theta^{(1)}_{12} = -0.52$
$\Theta^{(1)}_{22} = 0.08$
$\Theta^{(2)}_{10} = 0.13$
$\Theta^{(2)}_{20} = 0.48$
$\Theta^{(2)}_{11} = -0.69$
$\Theta^{(2)}_{21} = -0.73$
$\Theta^{(2)}_{12} = -0.70$
$\Theta^{(2)}_{22} = 0.11$
$\Theta^{(3)}_{10} = 0.55$
$\Theta^{(3)}_{11} = 0.76$
$\Theta^{(3)}_{12} = -0.25$

- Atualização dos pesos:

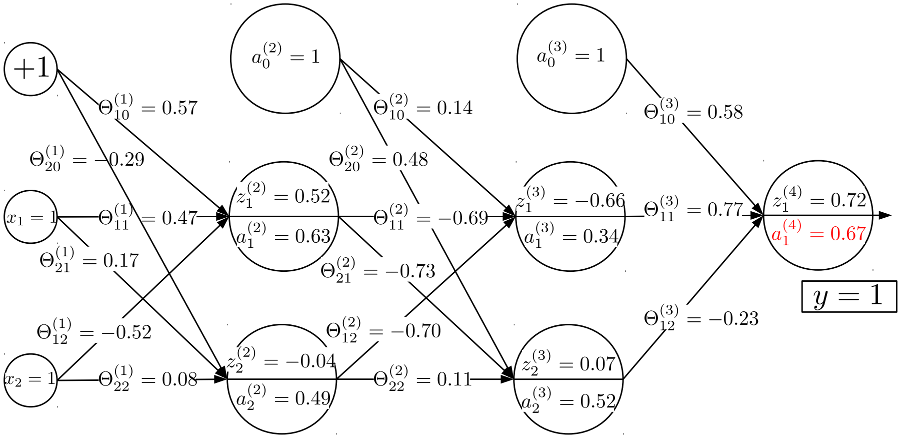
$$\Theta_{ij} := \Theta_{ij} - \alpha D_{ij}.$$

- No exemplo a seguir usaremos $\alpha = 0.1$.



$$\Theta^{(1)}_{21} := \Theta^{(1)}_{21} - \alpha D^{(1)}_{21} = 0.17 - 0.1 \cdot 0.01 = 0.17$$
$$\Theta^{(2)}_{10} := \Theta^{(2)}_{10} - \alpha D^{(2)}_{10} = 0.13 - 0.1 \cdot (-0.06) = 0.14$$
$$\Theta^{(3)}_{12} := \Theta^{(3)}_{12} - \alpha D^{(3)}_{12} = -0.25 - 0.1 \cdot (-0.18) = -0.23$$

...


+1
$a_0^{(2)} = 1$
$a_0^{(3)} = 1$
$\Theta_{10}^{(1)} = 0.57$
$\Theta_{20}^{(1)} = -0.29$
$x_1 = 1$
$\Theta_{11}^{(1)} = 0.47$
$\Theta_{21}^{(1)} = 0.17$
$\Theta_{12}^{(1)} = -0.52$
$x_2 = 1$
$\Theta_{22}^{(1)} = 0.08$
$z_1^{(2)} = 0.52$
$a_1^{(2)} = 0.63$
$z_2^{(2)} = -0.04$
$a_2^{(2)} = 0.49$
$\Theta_{10}^{(2)} = 0.14$
$\Theta_{20}^{(2)} = 0.48$
$\Theta_{11}^{(2)} = -0.69$
$\Theta_{21}^{(2)} = -0.73$
$\Theta_{12}^{(2)} = -0.70$
$\Theta_{22}^{(2)} = 0.11$
$z_1^{(3)} = -0.66$
$a_1^{(3)} = 0.34$
$z_2^{(3)} = 0.07$
$a_2^{(3)} = 0.52$
$\Theta_{10}^{(3)} = 0.58$
$\Theta_{11}^{(3)} = 0.77$
$\Theta_{12}^{(3)} = -0.23$
$z_1^{(4)} = 0.72$
$a_1^{(4)} = 0.67$
$y = 1$

Resumo

- $\delta_j^{(l)}$: "erro" do nó j na camada l.
- Na unidade de saída:
$$\delta_i^{(L)} = a_i^{(L)} - y.$$
- Para as demais camadas:
$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{ji}^{(l)} \delta_j^{(l+1)} \right) a_i^{(l)} (1 - a_i^{(l)}).$$
- Vetorização:
$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)}),$$
em que $.*$ é o produto ponto-a-ponto.
- Finalmente:
$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.}$$

Resumo

- $\delta_j^{(l)}$: "erro" do nó j na camada l.
- Na unidade de saída:
$$\delta_i^{(L)} = a_i^{(L)} - y.$$
- Para as demais camadas:
$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{ji}^{(l)} \delta_j^{(l+1)} \right) a_i^{(l)} (1 - a_i^{(l)}).$$
- Vetorização:
$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)}),$$
em que $.*$ é o produto ponto-a-ponto.
- Finalmente:
$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.}$$

Resumo

- $\delta_j^{(l)}$: "erro" do nó j na camada l.
- Na unidade de saída:

$$\delta_i^{(L)} = a_i^{(L)} - y.$$

- Para as demais camadas:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{ji}^{(l)} \delta_j^{(l+1)} \right) a_i^{(l)} (1 - a_i^{(l)}).$$

- Vetorização:

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)}),$$

em que $.*$ é o produto ponto-a-ponto.

- Finalmente:

$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.}$$

Resumo

- $\delta_j^{(l)}$: "erro" do nó j na camada l.
- Na unidade de saída:

$$\delta_i^{(L)} = a_i^{(L)} - y.$$

- Para as demais camadas:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{ji}^{(l)} \delta_j^{(l+1)} \right) a_i^{(l)} (1 - a_i^{(l)}).$$

- Vetorização:

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)}),$$

em que $.*$ é o produto ponto-a-ponto.

- Finalmente:

$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.}$$

Resumo

- $\delta_j^{(l)}$: "erro" do nó j na camada l.
- Na unidade de saída:
$$\delta_i^{(L)} = a_i^{(L)} - y.$$
- Para as demais camadas:
$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{ji}^{(l)} \delta_j^{(l+1)} \right) a_i^{(l)} (1 - a_i^{(l)}).$$
- Vetorização:
$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* \, a^{(l)} .* \, (1 - a^{(l)}),$$
em que $.*$ é o produto ponto-a-ponto.
- Finalmente:
$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}.}$$

Outline

$$\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$$

$$\boxed{\begin{array}{l} \Delta_{ij}^{(l)} := 0 \qquad \forall l, i, j \\ \text{Para } i = 1 \text{ até } m \\ \qquad a^{(1)} := x^{(i)} \\ \qquad \text{Calcular } a^{(l)} \text{ para } l = 2, 3, \cdots, L \ (\textit{forward}) \\ \qquad \delta^{(L)} := a^{(L)} - y^{(i)} \\ \qquad \text{Calcular } \delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)} \\ \qquad \Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \end{array}}$$

- NOTA: A última linha pode ser vetorizada por $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$.

$\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$

$\Delta_{ij}^{(l)} := 0 \qquad \forall l, i, j$
Para $i = 1$ até m
$\quad a^{(1)} := x^{(i)}$
$\quad$ Calcular $a^{(l)}$ para $l = 2, 3, \cdots, L$ (*forward*)
$\quad \delta^{(L)} := a^{(L)} - y^{(i)}$
$\quad$ Calcular $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$
$\quad \Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

- NOTA: A última linha pode ser vetorizada por $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$.

As derivadas parciais são obtidas pelo algoritmo a seguir:

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

- NOTA 1: Essa mesma derivada pode ser usada em algoritmos de otimização mais avançados.
- NOTA 2: Estes algoritmos normalmente exigem que a entrada seja um vetor. Para isso basta transformar a matriz D em um vetor.

As derivadas parciais são obtidas pelo algoritmo a seguir:

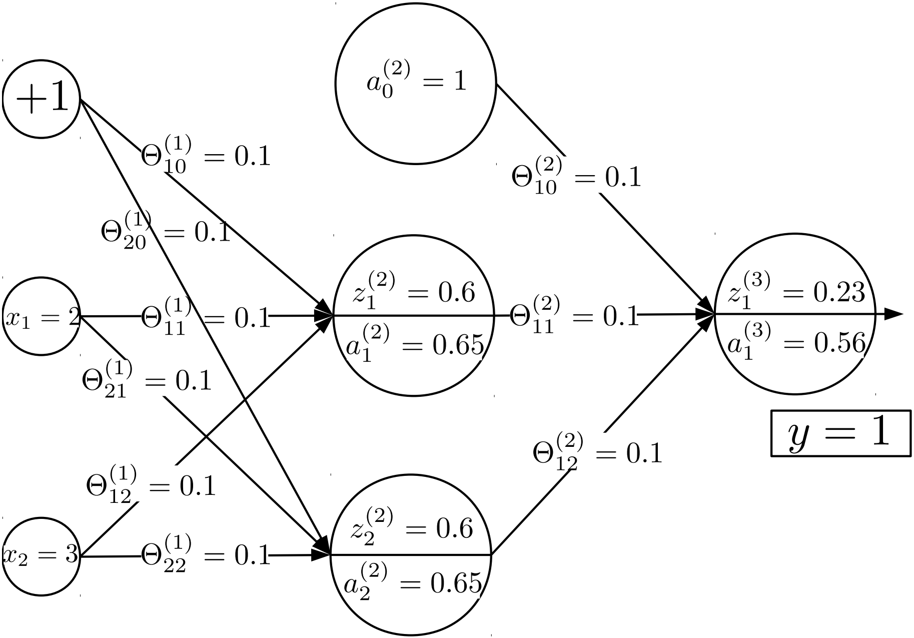
$$
\boxed{
\begin{array}{ll}
D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{se } j \neq 0 \\
D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} & \text{se } j = 0 \\
\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} &
\end{array}
}
$$

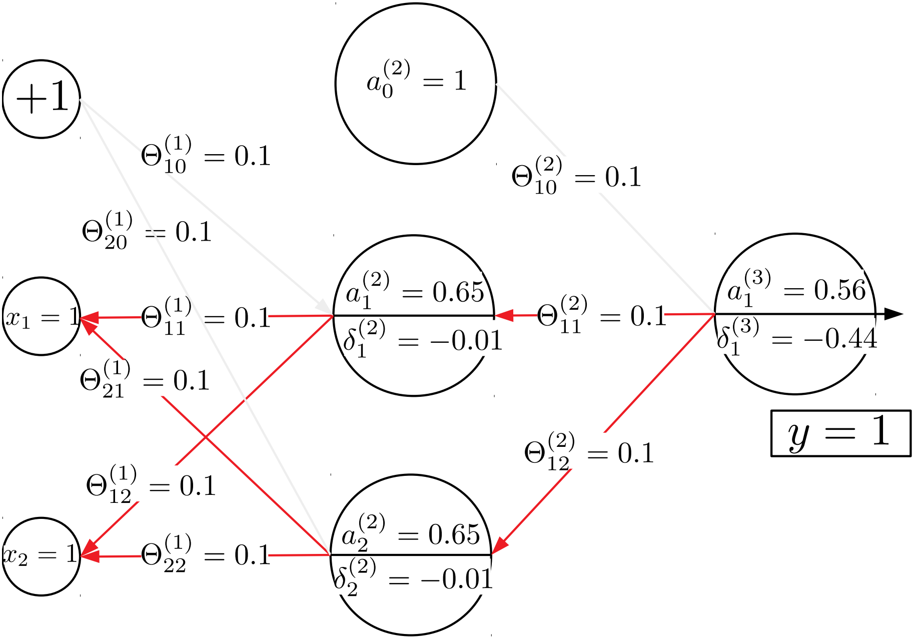
- NOTA 1: Essa mesma derivada pode ser usada em algoritmos de otimização mais avançados.
- NOTA 2: Estes algoritmos normalmente exigem que a entrada seja um vetor. Para isso basta transformar a matriz D em um vetor.

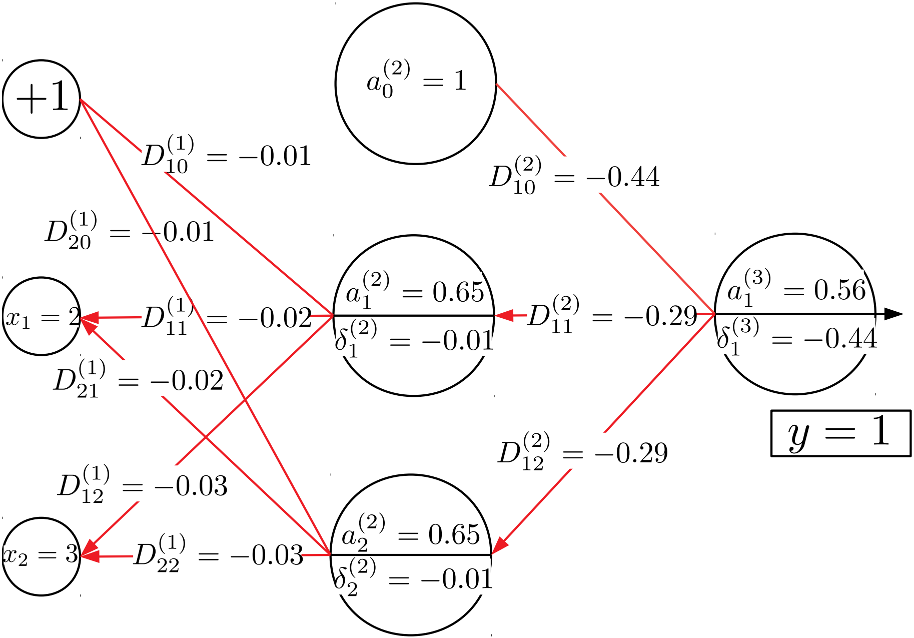
Outline

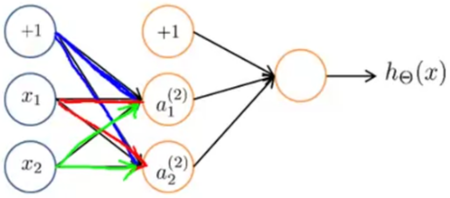
- Podemos inicializar todos os parâmetros com 0, como na regressão linear/logística?
- Ou, mais geral ainda, com um mesmo valor qualquer?

- Podemos inicializar todos os parâmetros com 0, como na regressão linear/logística?
- Ou, mais geral ainda, com um mesmo valor qualquer?

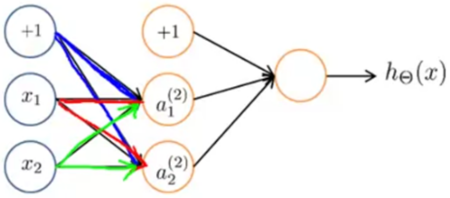

$+1$
$a_0^{(2)} = 1$
$\Theta_{10}^{(1)} = 0.1$
$\Theta_{20}^{(1)} = 0.1$
$\Theta_{10}^{(2)} = 0.1$
$x_1 = 2$
$\Theta_{11}^{(1)} = 0.1$
$\Theta_{21}^{(1)} = 0.1$
$z_1^{(2)} = 0.6$
$a_1^{(2)} = 0.65$
$\Theta_{11}^{(2)} = 0.1$
$z_1^{(3)} = 0.23$
$a_1^{(3)} = 0.56$
$y = 1$
$\Theta_{12}^{(2)} = 0.1$
$\Theta_{12}^{(1)} = 0.1$
$x_2 = 3$
$\Theta_{22}^{(1)} = 0.1$
$z_2^{(2)} = 0.6$
$a_2^{(2)} = 0.65$


+1
$a_0^{(2)} = 1$
$\Theta_{10}^{(1)} = 0.1$
$\Theta_{10}^{(2)} = 0.1$
$\Theta_{20}^{(1)} = 0.1$
$x_1 = 1$
$\Theta_{11}^{(1)} = 0.1$
$a_1^{(2)} = 0.65$
$\delta_1^{(2)} = -0.01$
$\Theta_{11}^{(2)} = 0.1$
$a_1^{(3)} = 0.56$
$\delta_1^{(3)} = -0.44$
$\Theta_{21}^{(1)} = 0.1$
$y = 1$
$\Theta_{12}^{(2)} = 0.1$
$\Theta_{12}^{(1)} = 0.1$
$x_2 = 1$
$\Theta_{22}^{(1)} = 0.1$
$a_2^{(2)} = 0.65$
$\delta_2^{(2)} = -0.01$


+1
$a_0^{(2)} = 1$
$D_{10}^{(1)} = -0.01$
$D_{10}^{(2)} = -0.44$
$D_{20}^{(1)} = -0.01$
$x_1 = 2$
$D_{11}^{(1)} = -0.02$
$a_1^{(2)} = 0.65$
$\delta_1^{(2)} = -0.01$
$D_{11}^{(2)} = -0.29$
$a_1^{(3)} = 0.56$
$\delta_1^{(3)} = -0.44$
$D_{21}^{(1)} = -0.02$
$y = 1$
$D_{12}^{(2)} = -0.29$
$D_{12}^{(1)} = -0.03$
$x_2 = 3$
$D_{22}^{(1)} = -0.03$
$a_2^{(2)} = 0.65$
$\delta_2^{(2)} = -0.01$



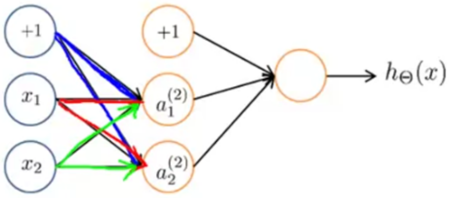
- Todos os pares com a mesma cor na figura terão sempre o mesmo valor em cada iteração.
- Sempre teremos então
$$a_1^{(2)} = a_2^{(2)}.$$
- Equivalente a um único nó na camada escondida.
- Ocorre sempre que inicializamos com o mesmo valor.



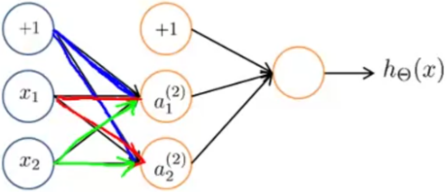
- Todos os pares com a mesma cor na figura terão sempre o mesmo valor em cada iteração.
- Sempre teremos então

$$a_1^{(2)} = a_2^{(2)}.$$

- Equivalente a um único nó na camada escondida.
- Ocorre sempre que inicializamos com o mesmo valor.



- Todos os pares com a mesma cor na figura terão sempre o mesmo valor em cada iteração.
- Sempre teremos então
$$a_1^{(2)} = a_2^{(2)}.$$
- Equivalente a um único nó na camada escondida.
- Ocorre sempre que inicializamos com o mesmo valor.



- Todos os pares com a mesma cor na figura terão sempre o mesmo valor em cada iteração.
- Sempre teremos então
$$a_1^{(2)} = a_2^{(2)}.$$
- Equivalente a um único nó na camada escondida.
- Ocorre sempre que inicializamos com o mesmo valor.

- Precisamos **quebrar a simetria**.
- Inicialização aleatória
$$-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon,$$
com $\epsilon \approx 0$.
- Versões avançadas - Xavier/He:
$$\Theta^{(l)} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_l + n_{l-1}}}\right).$$

Outline

- *Backpropagation* é um algoritmo complexo.
- Mesmo que $J(\theta)$ diminua, podemos ter um *bug*.
- Podemos checar o gradiente aproximando pela secante:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon},$$

em que um valor típico para ϵ é 10^{-4}.

- *Backpropagation* é um algoritmo complexo.
- Mesmo que $J(\theta)$ diminua, podemos ter um *bug*.
- Podemos checar o gradiente aproximando pela secante:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon},$$

em que um valor típico para ϵ é 10^{-4}.

- *Backpropagation* é um algoritmo complexo.
- Mesmo que $J(\theta)$ diminua, podemos ter um *bug*.
- Podemos checar o gradiente aproximando pela secante:

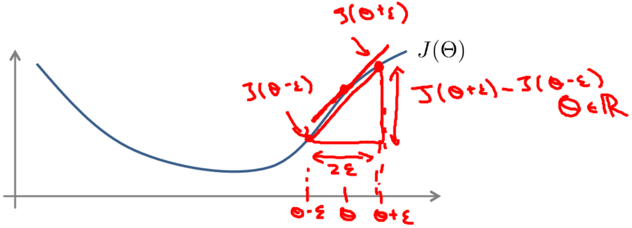
$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon},$$

em que um valor típico para ϵ é 10^{-4}.

- *Backpropagation* é um algoritmo complexo.
- Mesmo que $J(\theta)$ diminua, podemos ter um *bug*.
- Podemos checar o gradiente aproximando pela secante:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon},$$

em que um valor típico para ϵ é 10^{-4}.



- Na implementação, vetorizamos $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ e concatenamos, de modo a termos um vetor $\theta \in \mathbb{R}^n$ com TODOS os parâmetros da rede:

$$\theta = [\theta_1, \theta_2, \theta_3, \cdots, \theta_n]$$

O gradiente é então aproximado por:

$$\begin{aligned}
&\frac{\partial}{\partial\theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_3,\cdots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_3,\cdots,\theta_n)}{2\epsilon}\\
&\frac{\partial}{\partial\theta_2} J(\theta) \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_3,\cdots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_3,\cdots,\theta_n)}{2\epsilon}\\
&\vdots\\
&\frac{\partial}{\partial\theta_n} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\cdots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_3,\cdots,\theta_n-\epsilon)}{2\epsilon}.
\end{aligned}$$

- Certifica-se então de que este valor é próximo do obtido pelo gradiente descendente.
- Uma vez checado, esse gradiente numérico deve ser REMOVIDO do *backpropagation* pois é muito lento!

- Na implementação, vetorizamos $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ e concatenamos, de modo a termos um vetor $\theta \in \mathbb{R}^n$ com TODOS os parâmetros da rede:

$$\theta = [\theta_1, \theta_2, \theta_3, \cdots, \theta_n]$$

O gradiente é então aproximado por:

$$\begin{aligned}
&\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \cdots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \cdots, \theta_n)}{2\epsilon} \\
&\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \cdots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \cdots, \theta_n)}{2\epsilon} \\
&\vdots \\
&\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \cdots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \cdots, \theta_n-\epsilon)}{2\epsilon}.
\end{aligned}$$

- Certifica-se então de que este valor é próximo do obtido pelo gradiente descendente.
- Uma vez checado, esse gradiente numérico deve ser REMOVIDO do *backpropagation* pois é muito lento!

- Na implementação, vetorizamos $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ e concatenamos, de modo a termos um vetor $\theta \in \mathbb{R}^n$ com TODOS os parâmetros da rede:

$$\theta = [\theta_1, \theta_2, \theta_3, \cdots, \theta_n]$$

O gradiente é então aproximado por:

$$\begin{array}{l}
\frac{\partial}{\partial\theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_3,\cdots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_3,\cdots,\theta_n)}{2\epsilon} \\
\frac{\partial}{\partial\theta_2} J(\theta) \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_3,\cdots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_3,\cdots,\theta_n)}{2\epsilon} \\
\vdots \\
\frac{\partial}{\partial\theta_n} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\cdots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_3,\cdots,\theta_n-\epsilon)}{2\epsilon}.
\end{array}$$

- Certifica-se então de que este valor é próximo do obtido pelo gradiente descendente.
- Uma vez checado, esse gradiente numérico deve ser REMOVIDO do *backpropagation* pois é muito lento!

Outline

- 1º PASSO: Definir a arquitetura:
  - Na 1ª camada, o número de unidades é a dimensão da entrada $x^{(i)}$.
  - Na saída, é o número de classes.
  - Normalmente **uma** camada escondida (se tiver mais de uma recomenda-se que tenham o mesmo número de unidades).
  - Quanto mais unidade na camada escondida melhor. Limitante é o custo computacional. 3 ou 4 vezes a dimensão da entrada é usual.

- 1º PASSO: Definir a arquitetura:
  - Na 1ª camada, o número de unidades é a dimensão da entrada $x^{(i)}$.
  - Na saída, é o número de classes.
  - Normalmente **uma** camada escondida (se tiver mais de uma recomenda-se que tenham o mesmo número de unidades).
  - Quanto mais unidade na camada escondida melhor. Limitante é o custo computacional. 3 ou 4 vezes a dimensão da entrada é usual.

- 1º PASSO: Definir a arquitetura:
  - Na 1ª camada, o número de unidades é a dimensão da entrada $x^{(i)}$.
  - Na saída, é o número de classes.
  - Normalmente **uma** camada escondida (se tiver mais de uma recomenda-se que tenham o mesmo número de unidades).
  - Quanto mais unidade na camada escondida melhor. Limitante é o custo computacional. 3 ou 4 vezes a dimensão da entrada é usual.

- 1º PASSO: Definir a arquitetura:
  - Na 1ª camada, o número de unidades é a dimensão da entrada $x^{(i)}$.
  - Na saída, é o número de classes.
  - Normalmente **uma** camada escondida (se tiver mais de uma recomenda-se que tenham o mesmo número de unidades).
  - Quanto mais unidade na camada escondida melhor. Limitante é o custo computacional. 3 ou 4 vezes a dimensão da entrada é usual.

- 1º PASSO: Definir a arquitetura:
  - Na 1ª camada, o número de unidades é a dimensão da entrada $x^{(i)}$.
  - Na saída, é o número de classes.
  - Normalmente **uma** camada escondida (se tiver mais de uma recomenda-se que tenham o mesmo número de unidades).
  - Quanto mais unidade na camada escondida melhor. Limitante é o custo computacional. 3 ou 4 vezes a dimensão da entrada é usual.

Treinamento

1. Inicializar os pesos aleatoriamente.
2. *Forward propagation* para obter $h_\Theta(x^{(i)})$ para todo $x^{(i)}$.
3. Calcular função de custo $J(\Theta)$.
4. *Backpropagation* para calcular $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

   for i = 1:m

   Forward propagation e *backpropagation* usando $(x^{(i)}, y^{(i)})$
   (Obter as ativações $a^{(l)}$ e os deltas $\delta^{(l)}$ para $l = 2, \cdots, L$).
   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
   ...

   endfor

   ...

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

Treinamento

1. Inicializar os pesos aleatoriamente.
2. *Forward propagation* para obter $h_\Theta(x^{(i)})$ para todo $x^{(i)}$.
3. Calcular função de custo $J(\Theta)$.
4. *Backpropagation* para calcular $\frac{\partial}{\partial\Theta_{jk}^{(l)}}J(\Theta)$

   for i = 1:m

       Forward propagation e *backpropagation* usando $(x^{(i)}, y^{(i)})$
       (Obter as ativações $a^{(l)}$ e os deltas $\delta^{(l)}$ para $l = 2, \cdots, L$).
       $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
       ...

   endfor

   ...

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

Treinamento

1. Inicializar os pesos aleatoriamente.
2. *Forward propagation* para obter $h_\Theta(x^{(i)})$ para todo $x^{(i)}$.
3. Calcular função de custo $J(\Theta)$.
4. *Backpropagation* para calcular $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

   for i = 1:m

   Forward propagation e *backpropagation* usando $(x^{(i)}, y^{(i)})$
   (Obter as ativações $a^{(l)}$ e os deltas $\delta^{(l)}$ para $l = 2, \cdots, L$).
   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
   $\ldots$

   endfor

   $\ldots$

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

Treinamento

1. Inicializar os pesos aleatoriamente.
2. *Forward propagation* para obter $h_\Theta(x^{(i)})$ para todo $x^{(i)}$.
3. Calcular função de custo $J(\Theta)$.
4. *Backpropagation* para calcular $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

   for i = 1:m

     Forward propagation e *backpropagation* usando $(x^{(i)}, y^{(i)})$
     (Obter as ativações $a^{(l)}$ e os deltas $\delta^{(l)}$ para $l = 2, \cdots, L$).
     $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
     ...

   endfor

   ...

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

Treinamento

5. Comparar $\frac{\partial}{\partial\Theta_{jk}^{(l)}}$ do *backpropagation* com a estimativa numérica da checagem de gradiente.

6. Desabilitar a checagem de gradiente.

7. Usar gradiente descendente ou qualquer outro método avançado de otimização **em conjunto com** o *backpropagation* para minimizar $J(\Theta)$.

Treinamento

5. Comparar $\frac{\partial}{\partial \Theta_{jk}^{(l)}}$ do *backpropagation* com a estimativa numérica da checagem de gradiente.

6. Desabilitar a checagem de gradiente.

7. Usar gradiente descendente ou qualquer outro método avançado de otimização **em conjunto com** o *backpropagation* para minimizar $J(\Theta)$.

Treinamento

5. Comparar $\frac{\partial}{\partial \Theta_{jk}^{(l)}}$ do *backpropagation* com a estimativa numérica da checagem de gradiente.
6. Desabilitar a checagem de gradiente.
7. Usar gradiente descendente ou qualquer outro método avançado de otimização **em conjunto com** o *backpropagation* para minimizar $J(\Theta)$.

Observações

- NOTA 1: Existem versões mais avançadas que processam mais de um exemplo cada vez.
- NOTA 2: Função de custo das redes neurais em geral é **não convexa**. Algoritmo pode ficar preso em mínimos locais. Na prática, isso não é um grande problema!

Observações

- NOTA 1: Existem versões mais avançadas que processam mais de um exemplo cada vez.
- NOTA 2: Função de custo das redes neurais em geral é **não convexa**. Algoritmo pode ficar preso em mínimos locais. Na prática, isso não é um grande problema!

Outline

- É possível fazer regressão usando redes neurais?
- SIM! MAS.....em geral é "*overkill*"!
- Procedimento geral é idêntico, exceto por algumas pequenas mudanças:
  - Normalmente a função de ativação é a $ReLU(x) = \max(0, x)$.
  - Função de custo usual é o erro quadrático médio.

- É possível fazer regressão usando redes neurais?
- SIM! MAS.....em geral é "*overkill*"!
- Procedimento geral é idêntico, exceto por algumas pequenas mudanças:
  - Normalmente a função de ativação é a $ReLU(x) = \max(0, x)$.
  - Função de custo usual é o erro quadrático médio.

- É possível fazer regressão usando redes neurais?
- SIM! MAS.....em geral é "*overkill*"!
- Procedimento geral é idêntico, exceto por algumas pequenas mudanças:
  - Normalmente a função de ativação é a $ReLU(x) = \max(0, x)$.
  - Função de custo usual é o erro quadrático médio.

- É possível fazer regressão usando redes neurais?
- SIM! MAS.....em geral é "*overkill*"!
- Procedimento geral é idêntico, exceto por algumas pequenas mudanças:
  - Normalmente a função de ativação é a $ReLU(x) = \max(0, x)$.
  - Função de custo usual é o erro quadrático médio.

- É possível fazer regressão usando redes neurais?
- SIM! MAS.....em geral é "*overkill*"!
- Procedimento geral é idêntico, exceto por algumas pequenas mudanças:
  - Normalmente a função de ativação é a $ReLU(x) = \max(0, x)$.
  - Função de custo usual é o erro quadrático médio.