Advanced Institute for Artificial Intelligence

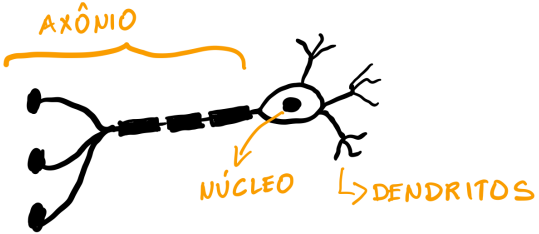
Redes Neurais com Perceptron Multicamadas

Advanced Institute for Artificial Intelligence – AI2

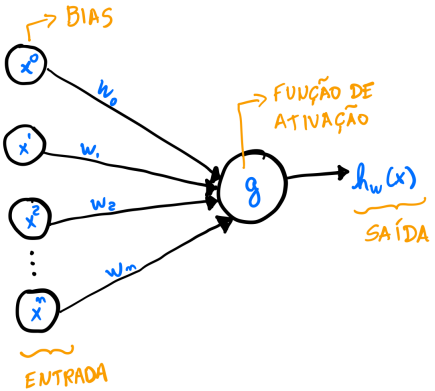
https://advancedinstitute.ai

Introdução

Redes Neurais foram amplamente utilizadas nos anos 80 e 90 e visam imitar o funcionamento do cérebro humano. Sua popularidade caiu no final dos anos 90, mas voltaram à cena com novas abordagens baseadas em aprendizado em profundidade. Mas como elas funcionam? Vamos dar uma olhada, primeiramente, na estrutura de um neurônio.



Matematicamente falando, podemos representar um neurônio da seguinte forma (Modelo de McCulloch-Pitts):



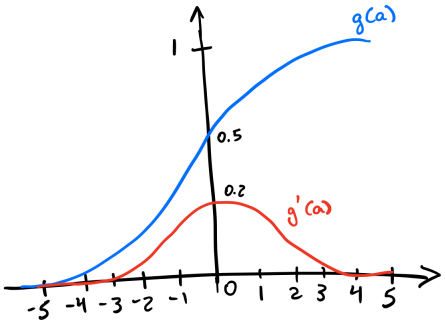
Alguns exemplos de funções de ativação:

- Função logística (sigmoide): $g(a) = \dfrac{1}{1 + e^{-a}}$, tal que $g(a) \in [0, 1]$.
- Função limiar (degrau):

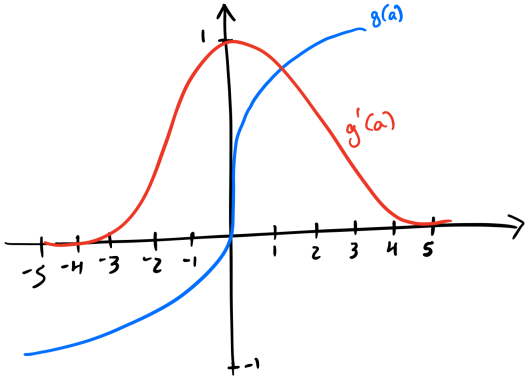
$$g_\theta(a) = \begin{cases} 1 \text{ caso } \boldsymbol{w}^T \boldsymbol{x} \geq \theta \\ 0 \text{ caso contrário.} \end{cases}$$

  tal que $g(a) \in \{0, 1\}$.
- Função tangente hiperbólica: $g(a) = 2\sigma(2a) - 1$, tal que $g(a) \in [-1, 1]$ e $\sigma(x)$ corresponde à função logística.

É desejável que uma função de ativação seja **derivável**. Como escolhê-la? Depende de sua aplicação.



Suponha $g(a) = \dfrac{1}{1+e^{-a}}$. Temos que $g'(a) = g(a)(1-g(a))$. Note que $g'(a)$ satura quando $a > 5$ ou $a < -5$. Ademais, $g'(a) < 1$, $\forall a$. Isso significa que, para redes com muitas camadas, o gradiente tende a **desaparecer** durante o treinamento.



Suponha $g(a) = 2\sigma(2a) - 1$. Temos que $g'(a) = 1 - g^2(a)$. Muito embora saturações aconteçam, $g'(a)$ atinge valores maiores, chegando, inclusive, ao máximo de 1 quando $a = 0$.

Perceptron

O Perceptron foi formalmente proposto por McCulloch e Pitts na década de 40 com o intuito de modelar matematicamente o neurônio humano. Muito embora ele tenha sido a base para muitos algoritmos, seu poder discriminativo é limitado, pois consegue aprender apenas hiperplanos como função de decisão.

Definição do problema: seja um conjunto de dados $\mathcal{X} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_z, y_z)\}$ tal que $\boldsymbol{x}_i \in \mathbb{R}^{n+1}$ corresponde ao dado de entrada e $y_i \in [-1, +1]$ denota o seu respectivo valor de saída. Temos, ainda, que $\mathcal{X}$ pode ser **particionado** da seguinte forma: $\mathcal{X} = \mathcal{X}^1 \cup \mathcal{X}^2$, em que $\mathcal{X}^1$ e $\mathcal{X}^2$ denotam os conjuntos de dados de **treinamento** e **teste**, respectivamente. Nosso objetivo é, dado o conjunto de treinamento, aprender uma função $h : \mathbb{R}^{n+1} \rightarrow \{-1, +1\}$ que consiga atribuir a classe correta à uma determinada amostra.

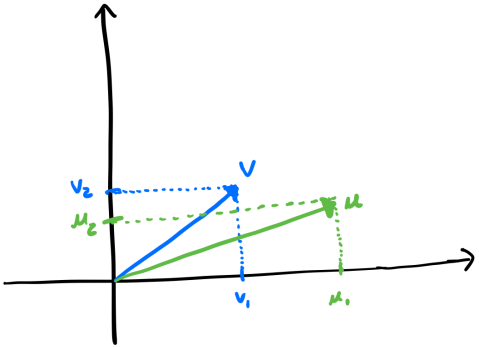
Vamos, agora, adaptar a função de ativação limiar para que possamos criar a nossa função hipótese:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} +1 \text{ caso } \boldsymbol{w}^T \boldsymbol{x} \geq \theta \\ -1 \text{ caso contrário.} \end{cases} \tag{1}$$

Para simplificar a notação, é usual trazermos θ para o lado esquerdo da equação e atribuirmos $w_0 = \theta$. Novamente, consideraremos $x^0 = 1$. Assim, temos a função hipótese atualizada como segue:

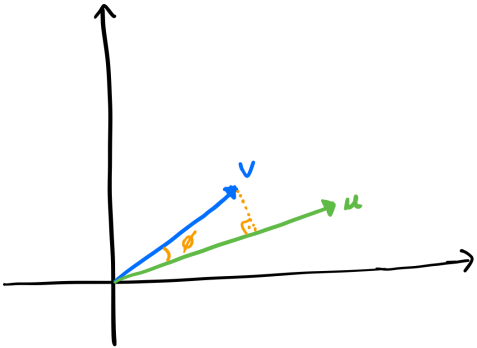
$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} +1 \text{ caso } \boldsymbol{w}^T \boldsymbol{x} \geq 0 \\ -1 \text{ caso contrário.} \end{cases} \tag{2}$$

Vamos, agora, revisitar alguns conceitos de Geometria Analítica. Suponha que tenhamos dois vetores $\boldsymbol{u} = [u_1\ u_2]$ e $\boldsymbol{v} = [v_1\ v_2]$. Geometricamente, podemos representá-los como segue:



Lembrando que $\|u\| = \sqrt{u_1^2 + u_2^2}$ denota o comprimento (magnitude) de u.

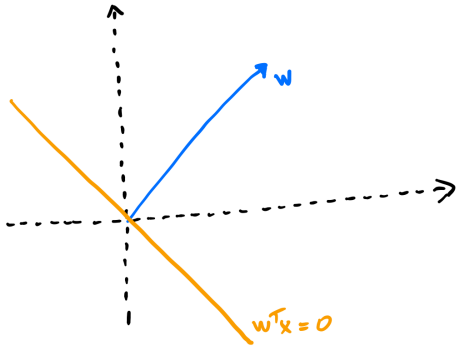
Temos, também, a definição de **projeção escalar** entre dois vetores, dada como segue:



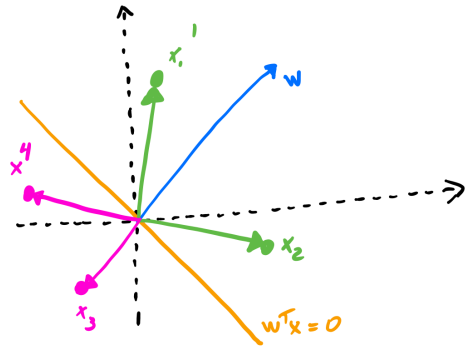
Podemos, então, representar a projeção escalar entre dois vetores da seguinte forma: $\boldsymbol{w}^T\boldsymbol{x} = \|\boldsymbol{w}\|.\cos\phi$. Lembrando que temos as seguintes situações:

- $\cos\phi > 0$ quando $0 < \phi < 90°$.
- $\cos\phi < 0$ quando $90° < \phi < 270°$.
- $\cos\phi = 0$ quando $\phi = 90°$ ou $\phi = 270°$

Voltando ao nosso problema inicial, temos que a equação $\boldsymbol{w}^T\boldsymbol{x} = 0$ define um hiperplano ortogonal ao vetor de pesos $\boldsymbol{w}$ deslocado de $-\theta$ (assumindo que $w_0 = -\theta$ e $x_0 = 1$). Vamos supor que $\theta = 0$, ou seja, o hiperplano passa pela origem.



Como aprender o conjunto de pesos $\boldsymbol{w}$? A ideia intuitiva é **mover** o vetor de pesos de tal forma que as amostras fiquem posicionadas corretamente no espaço de características. No exemplo abaixo, temos um conjunto de dados $\mathcal{X}^1 = \{(\boldsymbol{x}_1, +1), (\boldsymbol{x}_2, +1), (\boldsymbol{x}_3, -1), (\boldsymbol{x}_4, -1)\}$, de tal forma que o hiperplano $\boldsymbol{w}^T\boldsymbol{x} = 0$ já se encontra posicionado corretamente.



A regra de atualização dos pesos é dada pela seguinte fórmula:

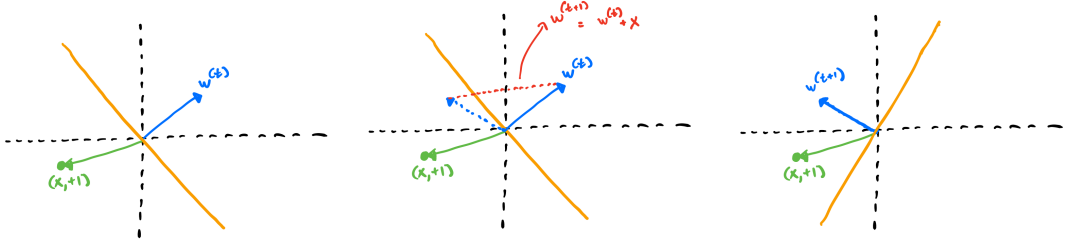
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha(y_i - h_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}_i))\boldsymbol{x}_i, \quad (3)$$

tal que $\forall i = 1, 2, \ldots, m$.

Mas como ele funciona na prática? Suponha uma amostra $\boldsymbol{x} \in \mathcal{X}$ tal que $y = +1$ e $h_{\boldsymbol{w}}(\boldsymbol{x}) = -1$. Para um valor de $\alpha = 0.5$, tempos que:

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} + \alpha(1 - (-1))\boldsymbol{x} \\ &= \boldsymbol{w}^{(t)} + 0.5(2)\boldsymbol{x} = \boldsymbol{w}^{(t)} + \boldsymbol{x}. \end{aligned} \tag{4}$$

Desta forma, o vetor de pesos $\boldsymbol{w}$ será rotacionado de modo que $\boldsymbol{x}$ seja positivo.



Predição | Atualização | Hiperplano aprendido

De maneira análoga, caso o rótulo de $\boldsymbol{x}$ seja negativo, ou seja, $y = -1$, temos que:

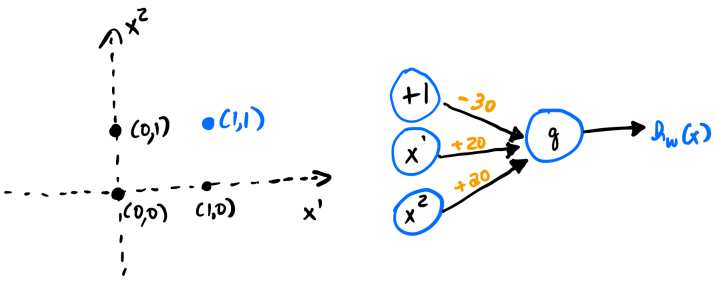
$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} + \alpha(-1 - (+1))\boldsymbol{x} \\ &= \boldsymbol{w}^{(t)} + 0.5(-2)\boldsymbol{x} = \boldsymbol{w}^{(t)} - \boldsymbol{x}. \end{aligned} \tag{5}$$

Assim, o vetor de pesos será rotacionado (por meio das projeções) para o outro lado. Para um conjunto de dados que seja **linearmente separável**, foi provado matematicamente que o algoritmo do Perceptron tem sua **convergência garantida**.

Como funciona o seu algoritmo? Vamos lá:

1. Atribua pesos aleatórios para $\boldsymbol{w}$.
2. Inicialize α.
3. $t = 0$.
4. Para cada amostra $\boldsymbol{x}_i \in \mathcal{X}^1$, faça:
    1. $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha(y_i - h_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}_i))\boldsymbol{x}_i$
5. Repita o passo 4 até algum critério de convergência for estabelecido.

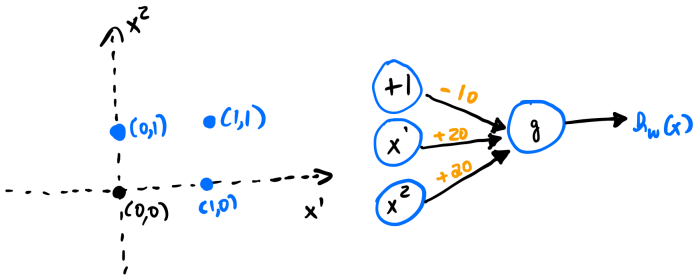
Vejamos alguns exemplos de funcionamento do Perceptron. Considere o problema de resolver a seguinte equação lógica: $y = x^1$ AND x^2. Para duas entradas, temos 2^2 possibilidades de amostras, isto é, o nosso conjunto de dados é composto pelos seguintes elementos: $\mathcal{X} = \{([0\ 0], 0), ([0\ 1], 0), ([1\ 0], 0), ([1\ 1], 1)\}$. Conseguimos encontrar o hiperplano separador?



Nossa função hipótese é dada por $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-30 + 20x^1 + 20x^2)$. Utilizando g como uma função logística, temos que:

x^1	x^2	$h_{\boldsymbol{w}}(\boldsymbol{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

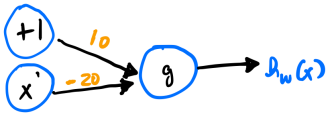
Considere, agora, o problema de resolver a seguinte equação lógica: $y = x^1$ OR x^2. Para duas entradas, temos 2^2 possibilidades de amostras, isto é, o nosso conjunto de dados é composto pelos seguintes elementos: $\mathcal{X} = \{([0\ 0], 0), ([0\ 1], 1), ([1\ 0], 1), ([1\ 1], 1)\}$. Conseguimos encontrar o hiperplano separador?



Nossa função hipótese é dada por $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-10 + 20x^1 + 20x^2)$. Utilizando g como uma função logística, temos que:

x^1	x^2	$h_{\boldsymbol{w}}(\boldsymbol{x})$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

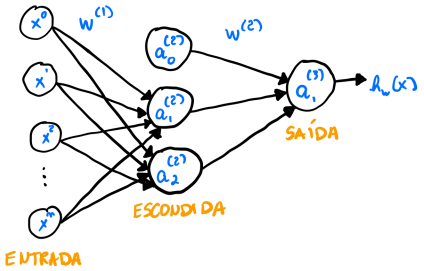
Considere, agora, o problema de resolver a seguinte equação lógica: $y = \text{NOT } x^1$. Agora temos apenas uma entrada, ou seja, x^1. Assim, o nosso conjunto de dados é composto pelos seguintes elementos: $\mathcal{X} = \{(0, 1), (1, 0)\}$. Conseguimos encontrar o hiperplano separador?



Nossa função hipótese é dada por $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(10 - 20x^1)$. Utilizando g como uma função logística, temos que:

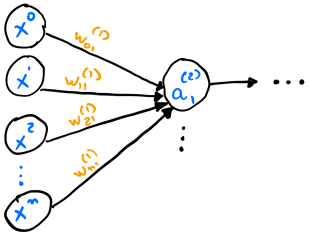
x^1	$h_{\boldsymbol{w}}(\boldsymbol{x})$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

O que seria então uma Rede Neural do tipo Perceptron Multicamadas (*Multilayer Perceptron* - MLP)? Basicamente, é um grupo de neurônios que, combinados, permitem o aprendizado de um número maior de funções de decisão.



Na ilustração acima, temos que $a_i^{(j)}$ denota o neurônio i da camada j, e $\boldsymbol{W}^{(l)}$ é a matriz de pesos que conecta as camadas l e $l+1$. Essa arquitetura é geralmente representada por $n{:}3{:}1$.

Considerando a rede neural anterior, vamos analisar uma situação mais específica.

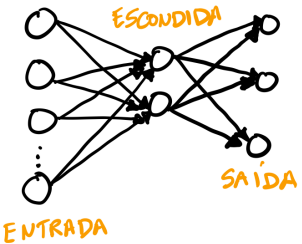


$$a_1^{(2)} = g(\underbrace{w_{01}^{(1)}x^0 + w_{11}^{(1)}x^1 + w_{02}^{(1)}x^2 + \ldots + w_{0n}^{(1)}x^n}_{\sum_{i=0}^{n} w_{i1}^{(1)}x^i = b_1^{(2)}})$$

A função de decisão final da rede neural é dada pela seguinte formulação:

$$h_{\boldsymbol{x}}(\boldsymbol{w}) = a_1^{(3)} = g\left(w_{01}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{21}^{(2)}a_2^{(2)}\right). \tag{6}$$

E no caso de problemas com múltiplas classes? Neste caso, para um problema com c classes, nossa camada de saída necessita de c neurônios.



Podemos utilizar a metodologia de codificação *one-hot* para representar cada neurônio de saída, em que $\boldsymbol{h_w}(\boldsymbol{x}) \in \mathbb{R}^3$:

$$h^1_{\boldsymbol{w}}(\boldsymbol{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad h^2_{\boldsymbol{w}}(\boldsymbol{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad h^3_{\boldsymbol{w}}(\boldsymbol{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Classe 1 Classe 2 Classe 3

O mesmo acontece com o rótulo y de cada amostra, que passa a ser agora um vetor $\boldsymbol{y} \in \mathbb{R}^3$.

Existem diversos algoritmos de treinamento para Redes Neurais MLP, em que o mais conhecido é chamado de **retropropagação**, do inglês *backpropagation*. Ele possui dois passos: (i) propagação para frente (*forward propagation*) e (ii) propagação para trás (*backpropagation*). Antes de estudarmos o seu funcionamento, vamos dar uma olhada na função de custo:

$$J(\boldsymbol{w}) = \frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{c} -y_i^k \log\Big(h_{\boldsymbol{w}}^k(\boldsymbol{x}_i)\Big) - (1-y_i^k)\log\Big(1-h_{\boldsymbol{w}}^k(\boldsymbol{x}_i)\Big)\right]. \tag{7}$$

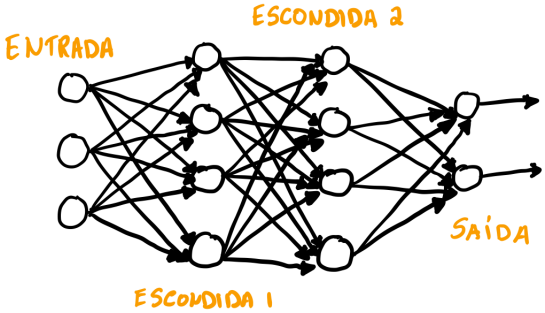
Fazendo uma analogia, a formulação acima contempla uma rede neural com c regressores logísticos caso tenhamos uma função de ativação logística na camada de saída.

Podemos utilizar, novamente, o gradiente descendente para otimizar a função de custo $J(\boldsymbol{w})$. No entanto, note que o problema passa a ser mais complexo, pois precisamos calcular as derivadas parciais com relação à todos os pesos da rede, ou seja:

$$\frac{\partial J(\boldsymbol{w})}{\partial w_{ij}^{(l)}}, \tag{8}$$

em que $l = 1, 2, \ldots, L-1$ tal que L denota a quantidade de camadas da rede neural.

Seja uma rede do tipo $3{:}4{:}4{:}2$ ilustrada abaixo. Suponha que tenhamos apenas uma amostra no conjunto de treinamento. Neste caso, o passo *forward* é dado pelas seguintes etapas:



1. $\boldsymbol{a}^{(1)} \leftarrow \boldsymbol{x}$
2. $\boldsymbol{b}^{(2)} \leftarrow \left(\boldsymbol{W}^{(1)}\right)^T \boldsymbol{a}^{(1)}$
3. $\boldsymbol{a}^{(2)} \leftarrow g\left(\boldsymbol{b}^{(2)}\right)$
4. $\boldsymbol{b}^{(3)} \leftarrow \left(\boldsymbol{W}^{(2)}\right)^T \boldsymbol{a}^{(2)}$
5. $\boldsymbol{a}^{(3)} \leftarrow g\left(\boldsymbol{b}^{(3)}\right)$
6. $\boldsymbol{b}^{(4)} \leftarrow \left(\boldsymbol{W}^{(3)}\right)^T \boldsymbol{a}^{(3)}$
7. $h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{a}^{(4)} \leftarrow g\left(\boldsymbol{b}^{(4)}\right)$

Para o passo de *backpropagation*, temos a definição de uma nova variável $\delta_j^{(l)}$ que denota um erro parcial acumulado no neurônio j da camada l. Este erro deve ser calculado de forma diferente para os neurônios de saída e das camada escondidas, como segue:

- Camada de saída ($l = 4$):

$$\begin{aligned} \delta_j^{(4)} &= a_j^{(4)} - y^j \\ &= h_{\boldsymbol{w}}^j(\boldsymbol{x}) - y^j \end{aligned} \tag{9}$$

Em notação vetorial, temos que $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \boldsymbol{y} = \boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}) - \boldsymbol{y}$.

- Camadas escondidas $l = \{2, 3\}$:
  - $\boldsymbol{\delta}^{(3)} = \left(\boldsymbol{W}^{(3)}\right)^T \boldsymbol{\delta}^{(4)} .* g'\left(\boldsymbol{b}^{(3)}\right)$
  - $\boldsymbol{\delta}^{(2)} = \left(\boldsymbol{W}^{(2)}\right)^T \boldsymbol{\delta}^{(3)} .* g'\left(\boldsymbol{b}^{(2)}\right)$

  Na prática, temos a seguinte formulação para propagação dos erros nas camadas intermediárias:

$$\boldsymbol{\delta}^{(l)} = \left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{\delta}^{(l+1)} .* g'\left(\boldsymbol{b}^{(l)}\right). \tag{10}$$

O nome **retropropagação** deve-se ao fato de o algoritmo “propagar para trás" o erro estimado em cada camada. As derivadas parciais podem ser calculadas como segue:

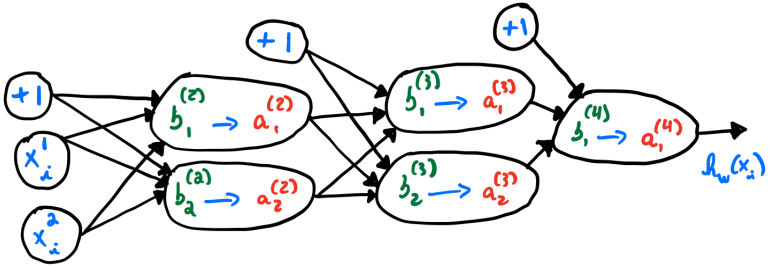
$$\frac{\partial J(\boldsymbol{w})}{\partial w_{ij}^{(l)}} = a_i^{(l)} \delta_j^{(l+1)}. \tag{11}$$

Note, então, que as derivadas parciais são calculadas com relação à todos os pesos da rede neural.

Segue algoritmo de retropropagação para treinamento de uma rede neural MLP.

1. Atribua pesos aleatórios para $w_{ij}^{(l)} \quad \forall l, i, j$
2. Execute os passos abaixo ate o critério de parada ter sido estabelecido: Laço das épocas
   1. $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ Variável utilizada para armazenar $\frac{\partial J(\boldsymbol{w})}{\partial w_{ij}^{(l)}}$
   2. Para cada amostra $\boldsymbol{x}_i \in \mathcal{X}^1$, faça:
      1. Execute o passo de *forward propagation* para calcular $\boldsymbol{a}^l, \ l = 2, 3, \ldots, L$
      2. Execute o passo de *backward propagation* para calcular:
      3. $\boldsymbol{\delta}^l, \ l = L, L-1, \ldots, 2$ Erro em cada neurônio
      4. $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_i^{(l)} \delta_j^{(l+1)}$ Derivadas parciais são acumuladas
   3. $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \quad \forall l, i, j$ Calcula o gradiente médio
   4. $w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha D_{ij}^{(l)} \quad \forall l, i, j$ Atualiza pesos com gradiente descendente
   5. Avalie a função de custo $J(\boldsymbol{w})$

Resumindo: passo *forward*.



Resumindo: passo *backward*.