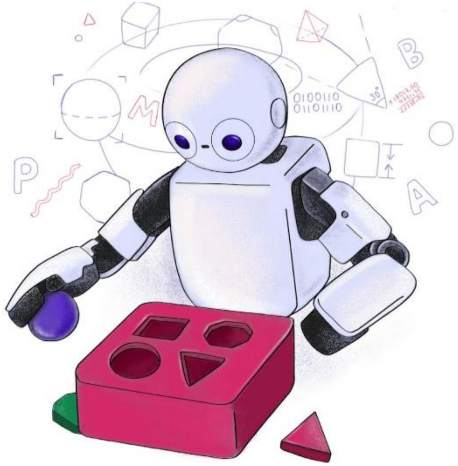
T320 - Introdução ao Aprendizado de Máquina II:
Redes Neurais Artificiais (Parte II)



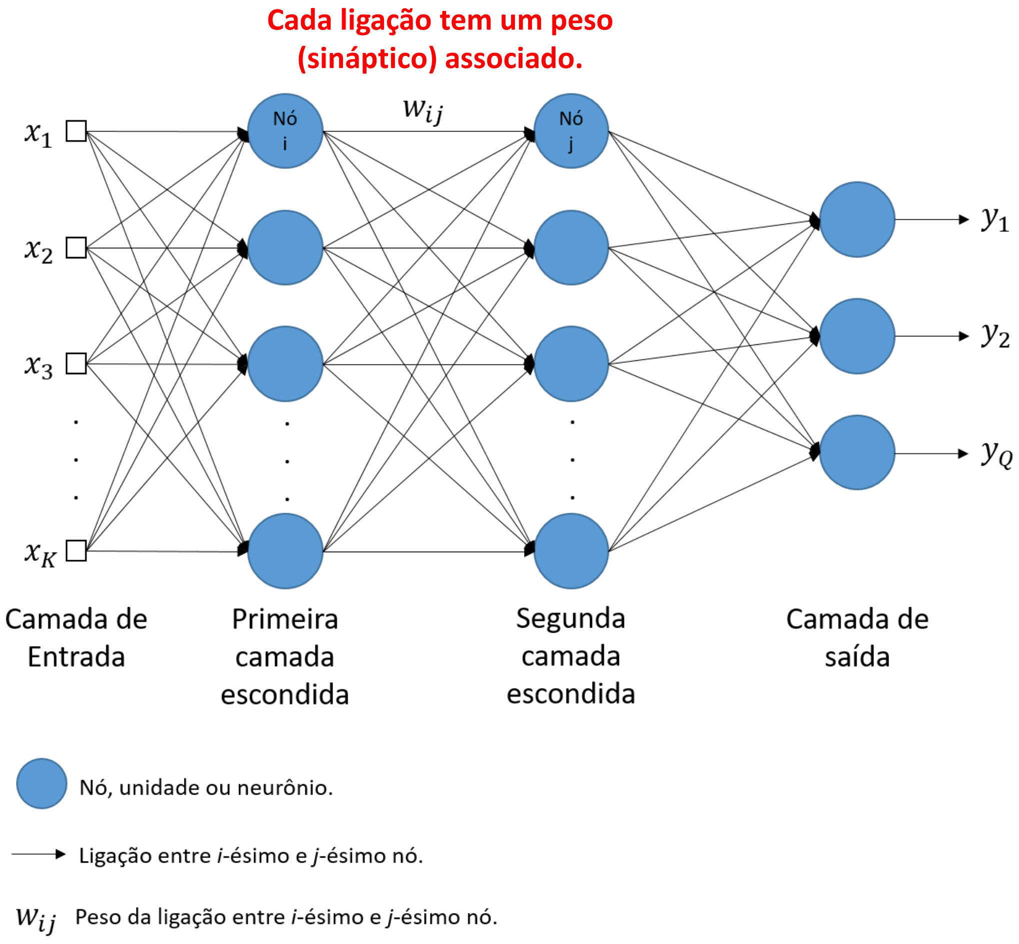
Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Recapitulando

- Fizemos uma analogia entre um neurônio e os modelos de McCulloch e Pitts e do Perceptron.
- Vimos a evolução do modelo de McCulloch e Pitts para o Perceptron.
- Aprendemos suas características, diferenças e como ambos funcionam.
- Verificamos que um Perceptron é semelhante ao regressor logístico.
- Constatamos que um ***único*** Perceptron não é capaz de separar classes não-lineares, como, por exemplo, o problema da lógica XOR.
- Porém, quando combinamos vários deles, conseguimos criar um separador não-linear.
- Neste tópico, veremos que esta união de Perceptrons origina o que chamamos de ***redes neurais artificiais (RNAs)***.

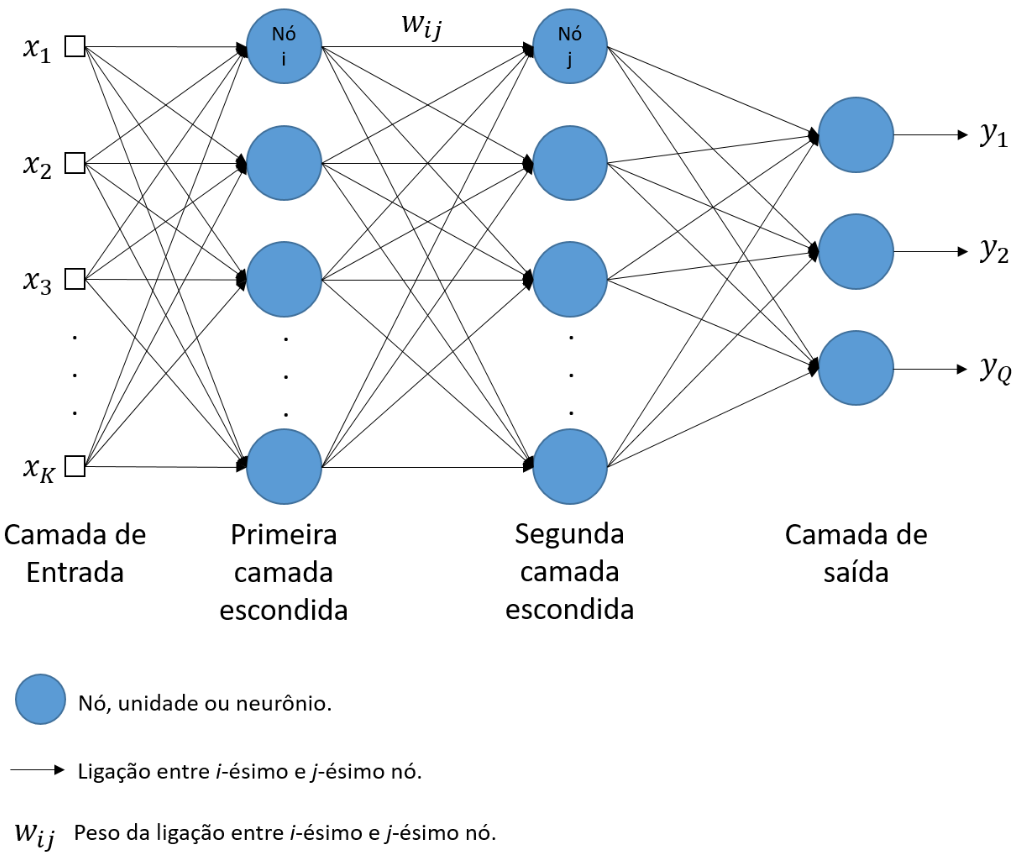
Perceptron de múltiplas camadas



- Uma ***rede neural*** nada mais é do que uma ***combinação de neurônios*** conectados entre si através de ***ligações direcionadas*** (ou seja, as conexões têm uma direção associada).
  - Neurônios também são chamados de ***nós*** ou ***unidades***.
  - ***Cada ligação*** entre nós ***possui um peso (sináptico) associado***.
- As ***propriedades da rede neural*** são determinadas por sua ***arquitetura***, i.e., como os neurônios estão conectados, quantidade neurônios e de camadas escondidas, função de ativação, etc.

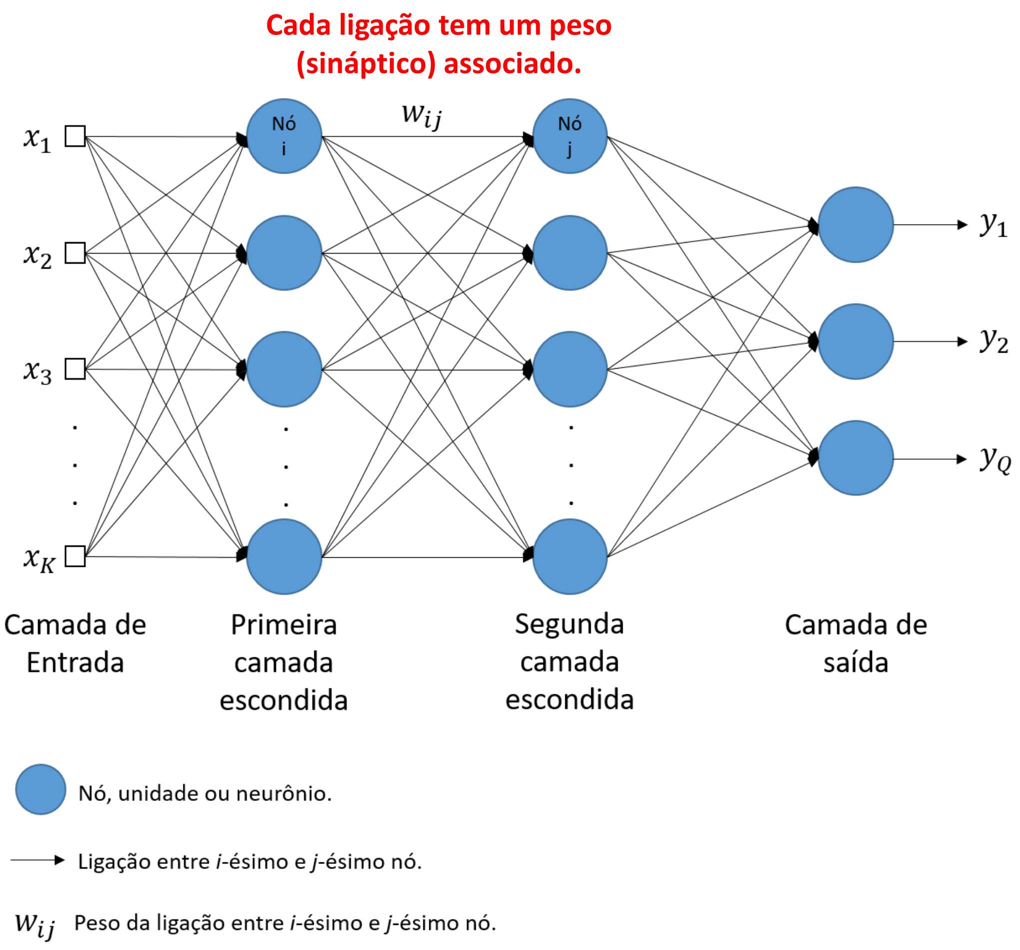
Perceptron de múltiplas camadas

Cada ligação tem um peso (sináptico) associado.



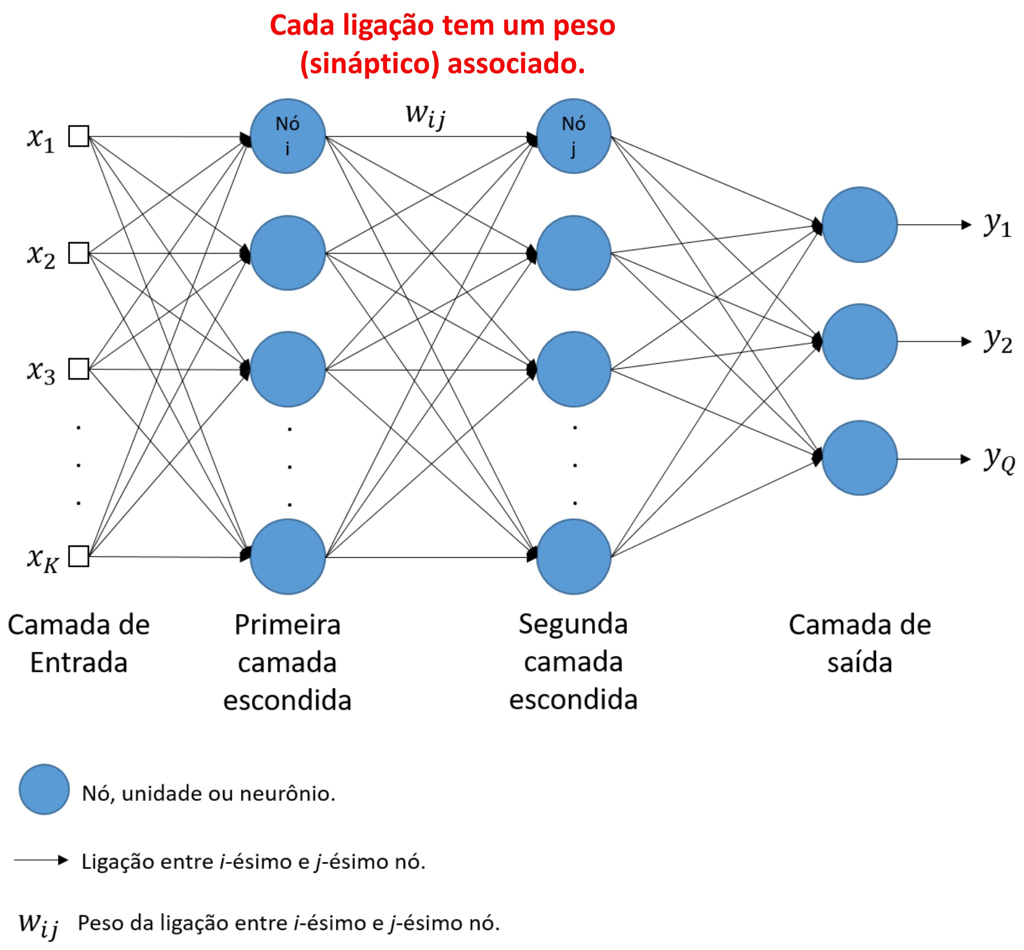
- Algumas das ***limitações dos perceptrons*** (e.g., classificação apenas de classes linearmente separáveis) podem ser ***superadas adicionando-se camadas intermediárias*** de ***perceptrons***.
- As camadas intermediárias são também chamadas de ***ocultas*** ou ***escondidas***.

Perceptron de múltiplas camadas



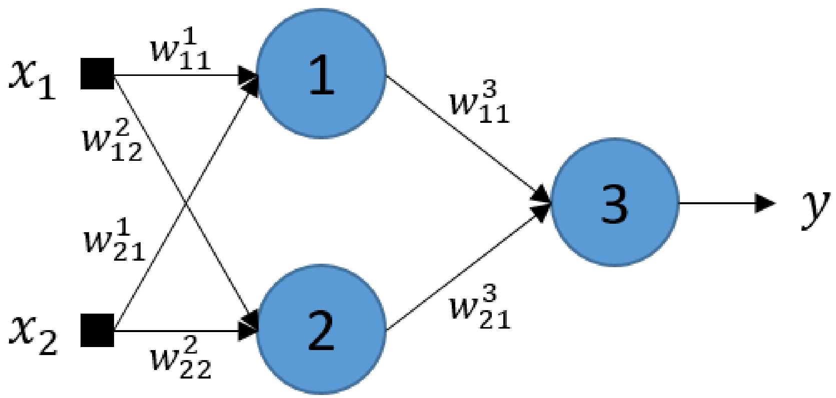
- A rede ao lado é do tipo ***densamente conectada*** e de ***alimentação direta***.
  - Cada uma das saídas de uma camada se conecta a todos os nós da camada seguinte através de pesos sinápticos.
  - Os dados fluem através da rede em uma única direção, da camada de entrada para a camada de saída, sem ciclos ou *loops* de retroalimentação.
- Essa rede é chamada de ***perceptron de múltiplas camadas*** (do inglês, *Multilayer Perceptron* - MLP) ou de ***rede densamente conectada*** (do inglês, *Dense Neural Network* - DNN).

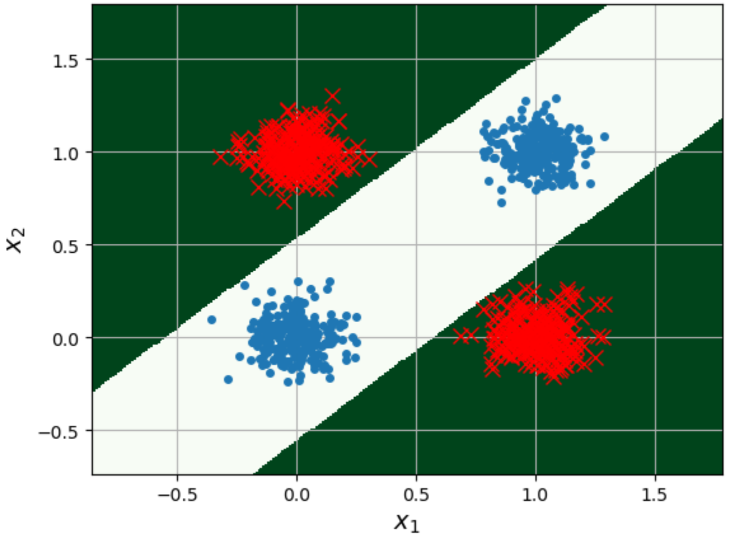
Perceptron de múltiplas camadas



- As RNAs são o coração do ***deep learning ou aprendizado profundo***.
- O termo "***profundo***" vem fato de que essas redes podem possuir muitas camadas ocultas.
- Em geral, quando uma RNA tem duas ou mais camadas ocultas, ela pode ser chamada de ***rede neural profunda*** (ou em inglês, *Deep Neural Network* - DNN).
- A rede MLP ao lado possui duas camadas ocultas e, portanto, poderia ser chamada de DNN.

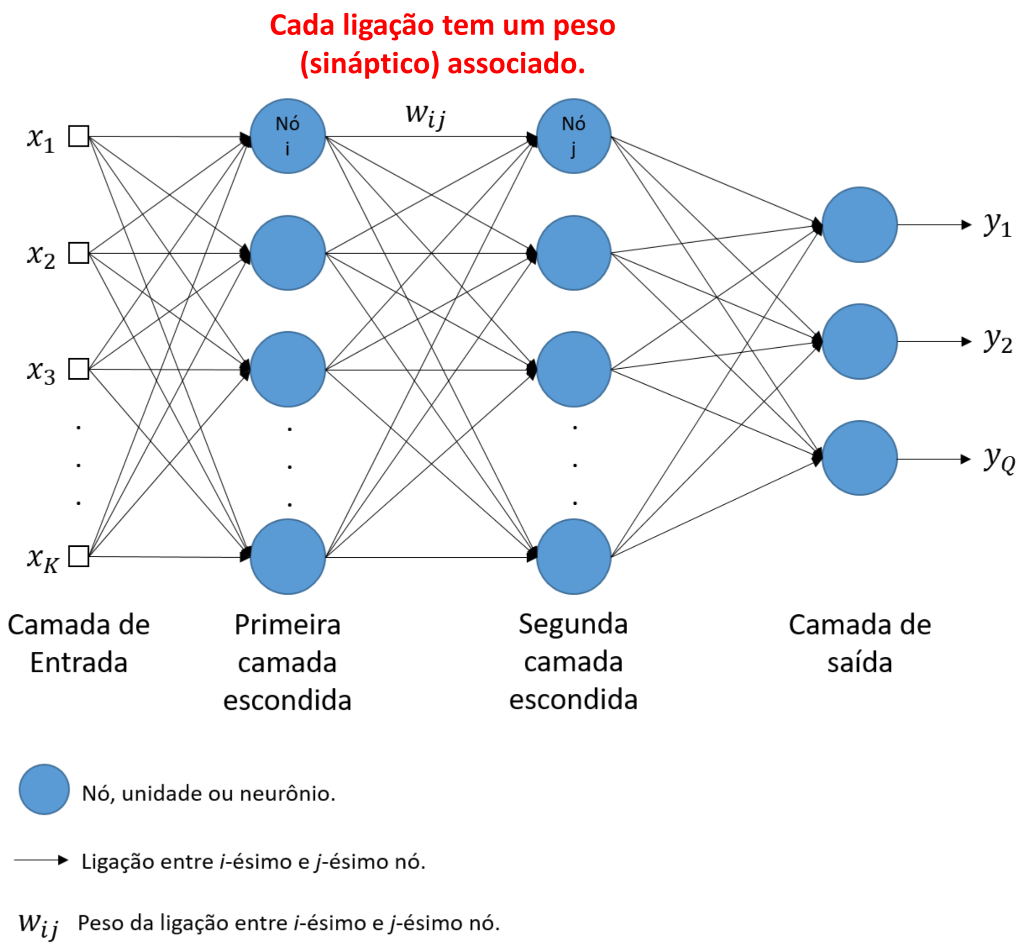
Perceptron de múltiplas camadas





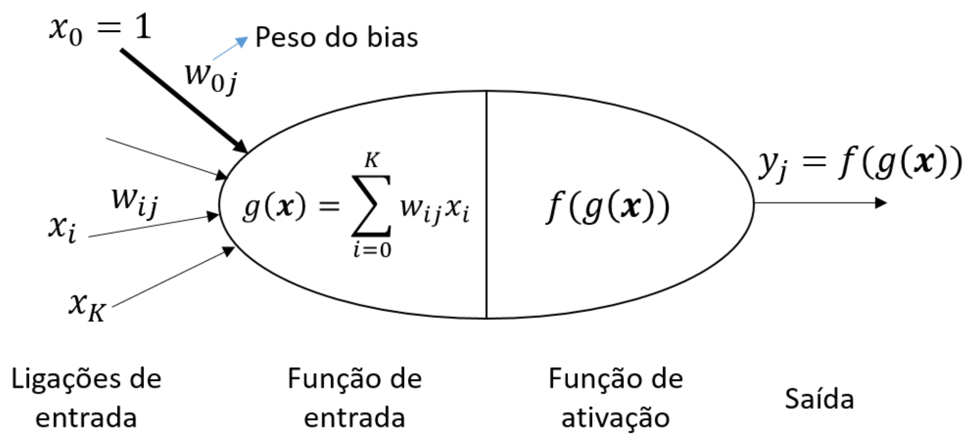
- Em particular, uma MLP com ***uma camada oculta com dois nós*** e ***uma camada de saída com um nó*** pode resolver o problema da lógica XOR.
- Lembrem-se que um único ***perceptron*** não é capaz de realizar essa tarefa.
- Os dois nós da camada oculta aprendem separadores lineares que são combinados para obter a separação não linear resultante.

Perceptron de múltiplas camadas



- A ***ligação*** do i-ésimo ***nó*** para o j-ésimo ***nó*** é feita através do ***peso*** w_{ij} e serve para ***propagar o sinal de ativação*** do i-ésimo ***nó*** para o j-ésimo ***nó***.
- O ***sinal de ativação*** é a saída do i-ésimo nó e é denotado por x_i.
- O valor do ***peso*** determina a ***força*** e o ***sinal*** da ***ligação***.
- A ligação pode ser ***excitatória*** ou ***inibitória*** dependendo do sinal do peso.

Perceptron de múltiplas camadas



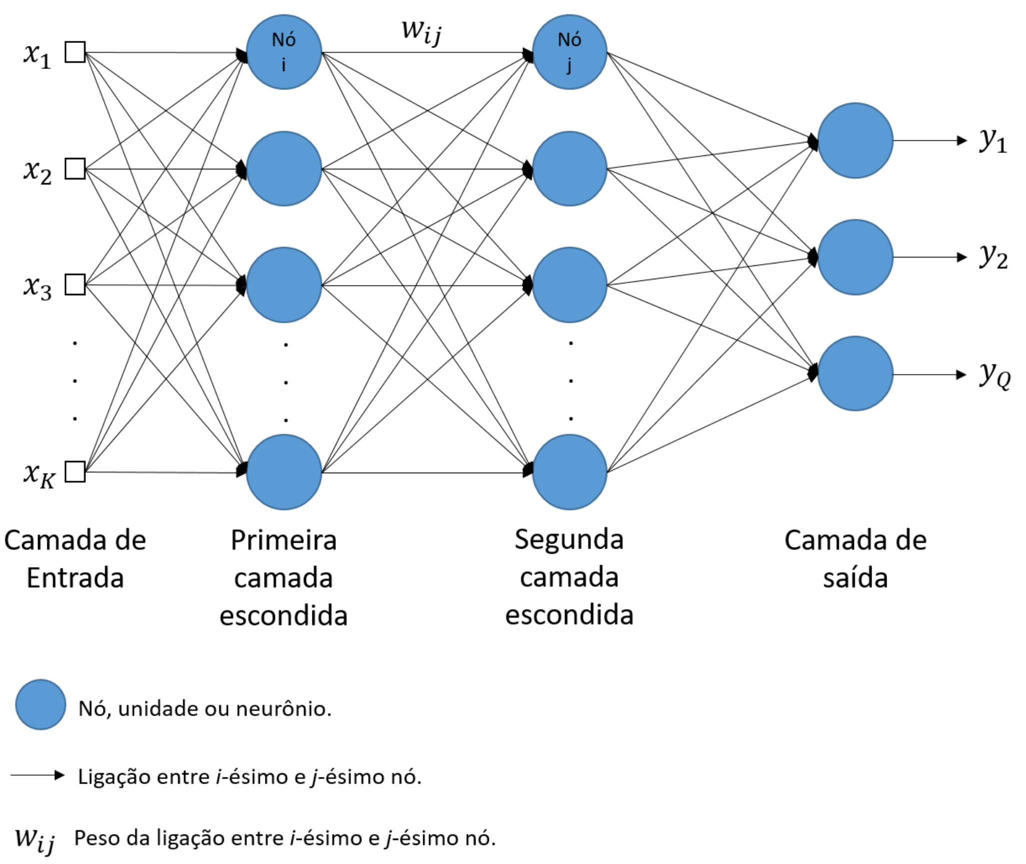
- Cada ***nó*** tem a entrada x_0 (i.e., o atributo de bias) sempre com valor igual a 1 e um peso associado w_{0j}, chamado de ***peso de bias***.
  - Ou seja, a entrada x_0 ***não está conectada a nenhum outro nó***.
- O j-ésimo ***nó*** calcula a ***soma ponderada*** de suas entradas, x_i

$$g(\boldsymbol{x}) = \sum_{i=0}^{K} w_{ij} x_i = \boldsymbol{w}^T \boldsymbol{x},$$

e, em seguida, aplica uma ***função de ativação*** (i.e., de limiar), $f(.)$, à soma para gerar sua saída

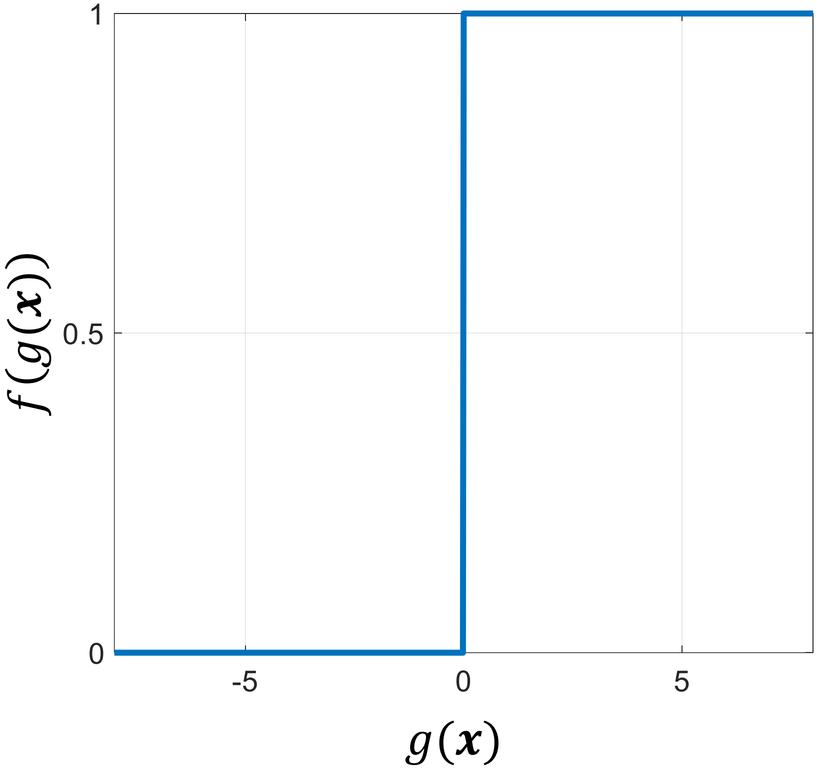
$$y_j = f(g(\boldsymbol{x})).$$

Perceptron de múltiplas camadas



- Existem vários tipos de ***funções de ativação*** que podem ser utilizadas pelos ***nós*** de uma rede neural.
- Cada camada pode usar funções de ativação diferentes.
- Porém, em geral, todos os nós de uma camada usam a mesma função de ativação.

Funções de ativação



- Devido a suas características, não se utiliza a ***função degrau*** como função de ativação em redes neurais.
  - Derivada sempre igual a zero, exceto na origem, onde ela é indeterminada.
- Até o surgimento das ***redes neurais profundas***, a regra era utilizar as ***funções logística*** ou ***tangente hiperbólica***, que são versões suavizadas da função degrau.
  - Essas funções ***são contínuas e possuem derivada definida e diferente de 0 em todos os pontos***.

Função logística

- A saída de um nó com ***função de ativação logística*** (ou sigmoide) tem a seguinte expressão

$$y_j = f(g(\boldsymbol{x})) = \frac{1}{1 + e^{-g(\boldsymbol{x})}},$$

onde $g(\boldsymbol{x})$ é a ***combinação linear das entradas do nó***.

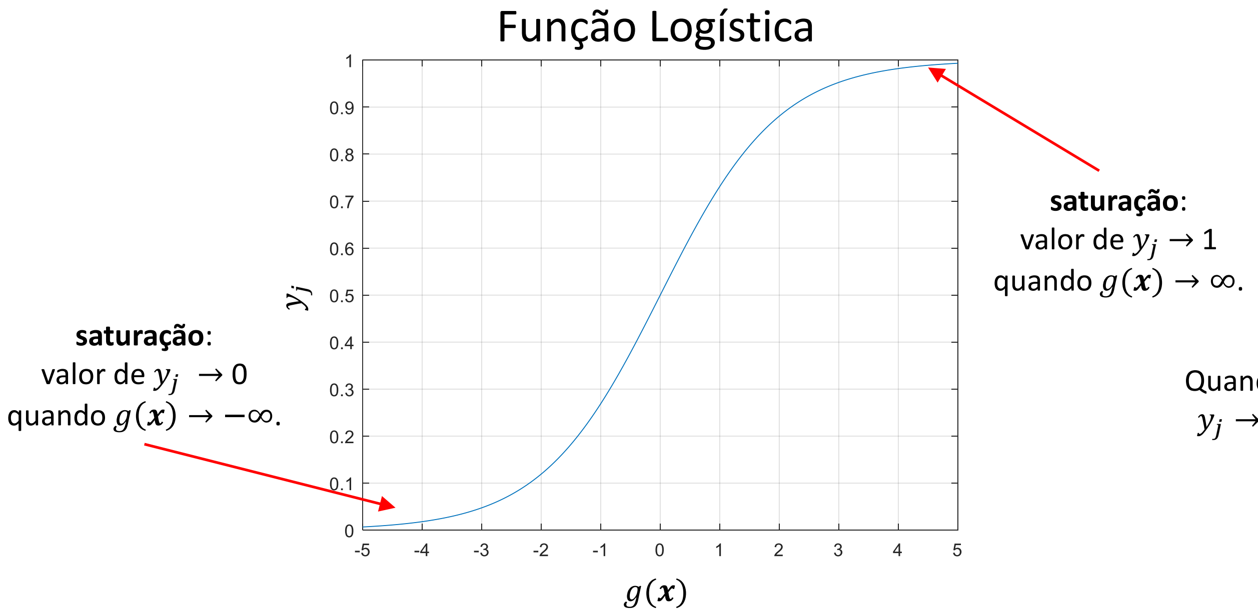
- Sua derivada é dada por

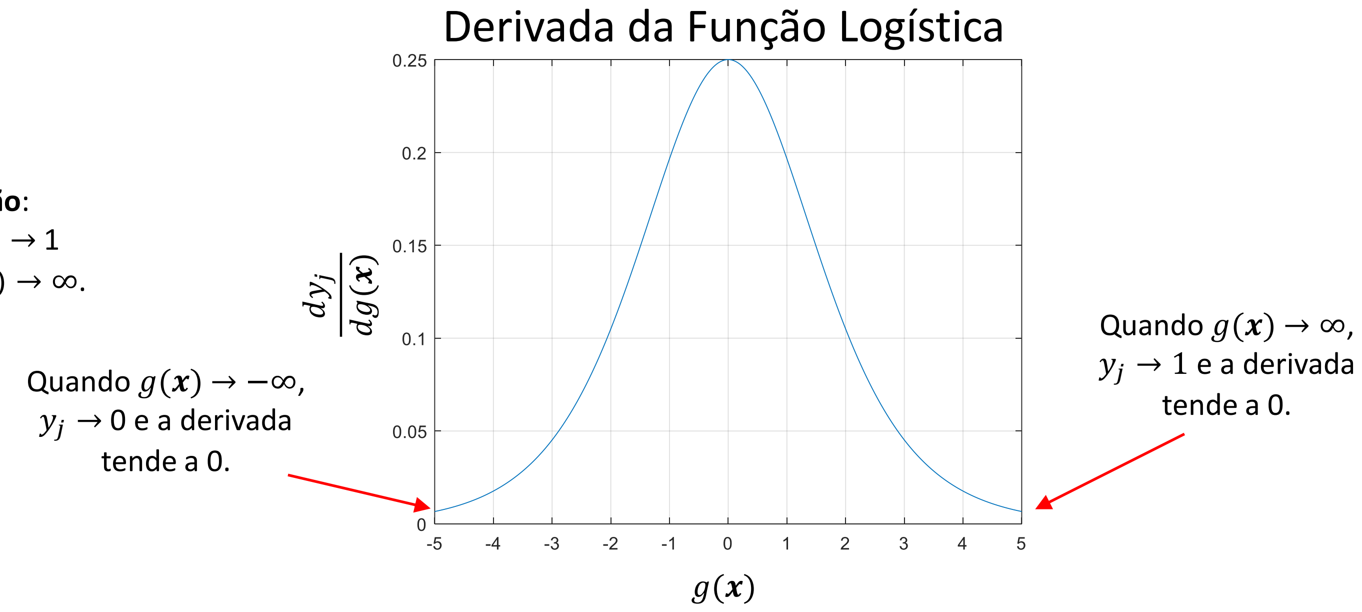
$$\frac{dy_j}{dg(\boldsymbol{x})} = \frac{df(g(\boldsymbol{x}))}{dg(\boldsymbol{x})} = y_j(1 - y_j) \geq 0.$$

- A derivada será importante durante o processo de aprendizado da rede neural.

Função logística e sua derivada

- Percebam que o valor da derivada ***sempre será menor do que 1***, ***sendo no máximo igual a 0.25 quando*** $g(\boldsymbol{x}) = 0$.





Função tangente hiperbólica

- A saída de um nó com ***função de ativação tangente hiperbólica*** tem sua expressão dada por

$$y_j = f(g(\boldsymbol{x})) = \tanh(g(\boldsymbol{x})) = \frac{e^{g(\boldsymbol{x})} - e^{-g(\boldsymbol{x})}}{e^{g(\boldsymbol{x})} + e^{-g(\boldsymbol{x})}}.$$

onde $g(\boldsymbol{x})$ é a ***combinação linear das entradas do nó***.

- Sua derivada é dada por

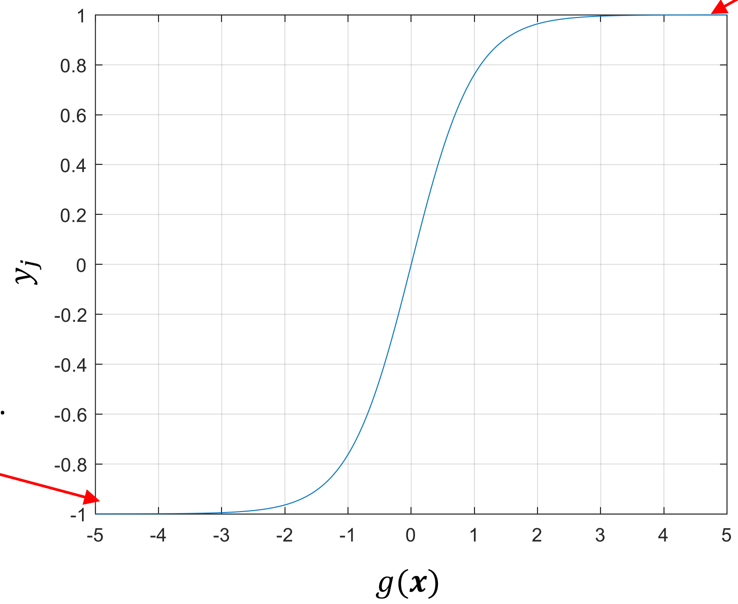
$$\frac{dy_j}{dg(\boldsymbol{x})} = \frac{df(g(\boldsymbol{x}))}{dg(\boldsymbol{x})} = 1 - \tanh^2(g(\boldsymbol{x})) \geq 0.$$

Função tangente hiperbólica e sua derivada

- A derivada é no máximo igual a 1 ***exatamente*** quando ***quando*** $g(\boldsymbol{x}) = 0$, sendo menor do que 1 para todos os outros valores de $g(\boldsymbol{x})$.

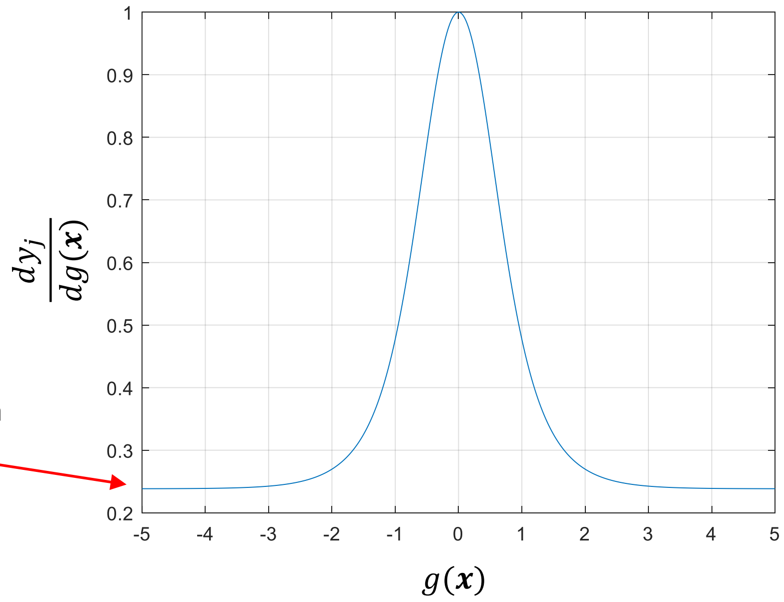
saturação:
valor de $y_j \to 1$
quando $g(\boldsymbol{x}) \to \infty$.

Função Tangente Hiperbólica



saturação:
valor de $y_j \to -1$
quando $g(\boldsymbol{x}) \to -\infty$.

Derivada da Tangente Hiperbólica



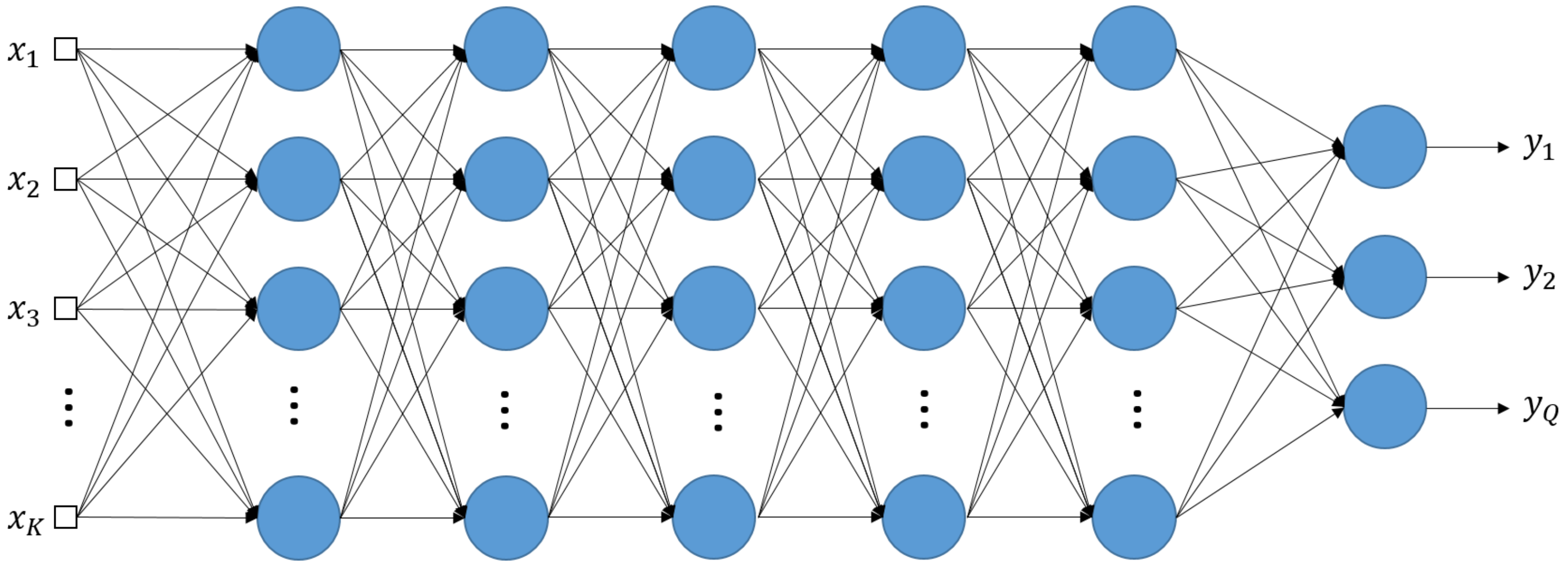
Quando $g(\boldsymbol{x}) \to -\infty$,
$y_j \to -1$ e a derivada
tende a 0.

Quando $g(\boldsymbol{x}) \to \infty$,
$y_j \to 1$ e a derivada
tende a 0.

Na sequência, veremos que esses valores de derivadas menores do que 1 causam um problema no aprendizado de redes com muitas camadas, i.e., redes profundas.

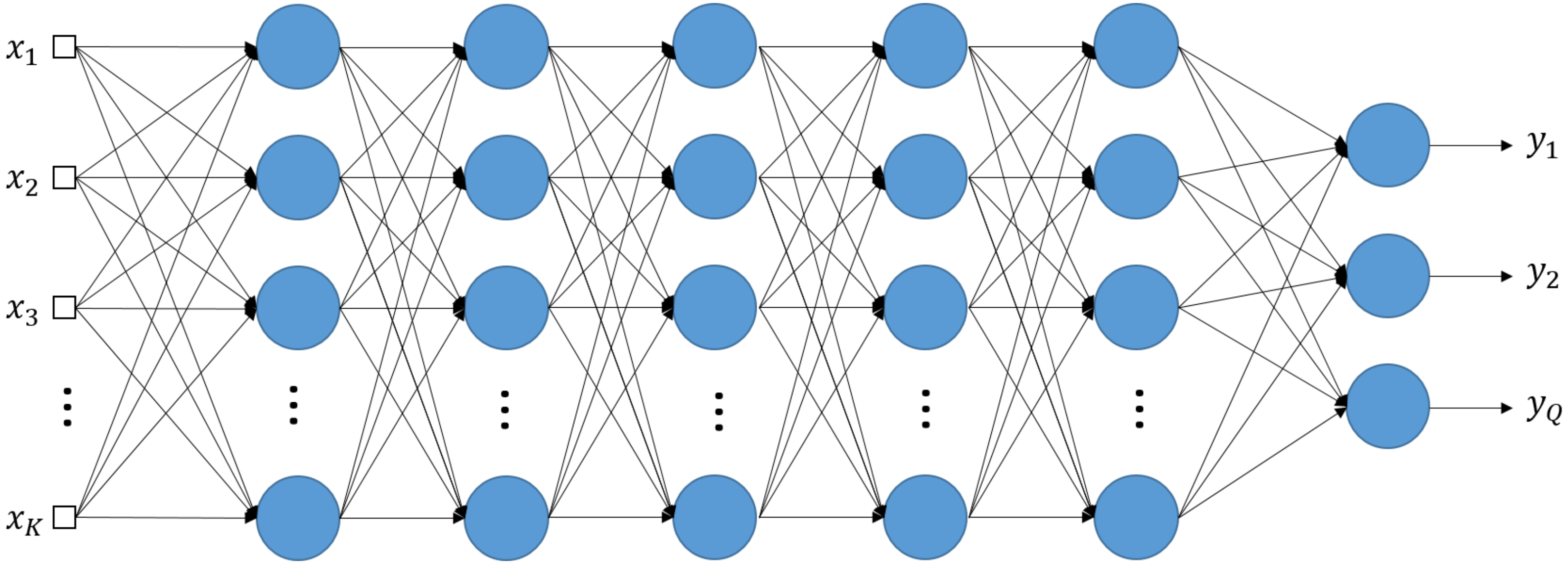
O problema da dissipação do gradiente

- É um problema encontrado quando treinamos ***redes neurais profundas***, ou seja, com muitas camadas ocultas, com ***métodos de aprendizado baseados no gradiente descendente*** e nós usando ***funções de ativação sigmoide ou tangente hiperbólica***.



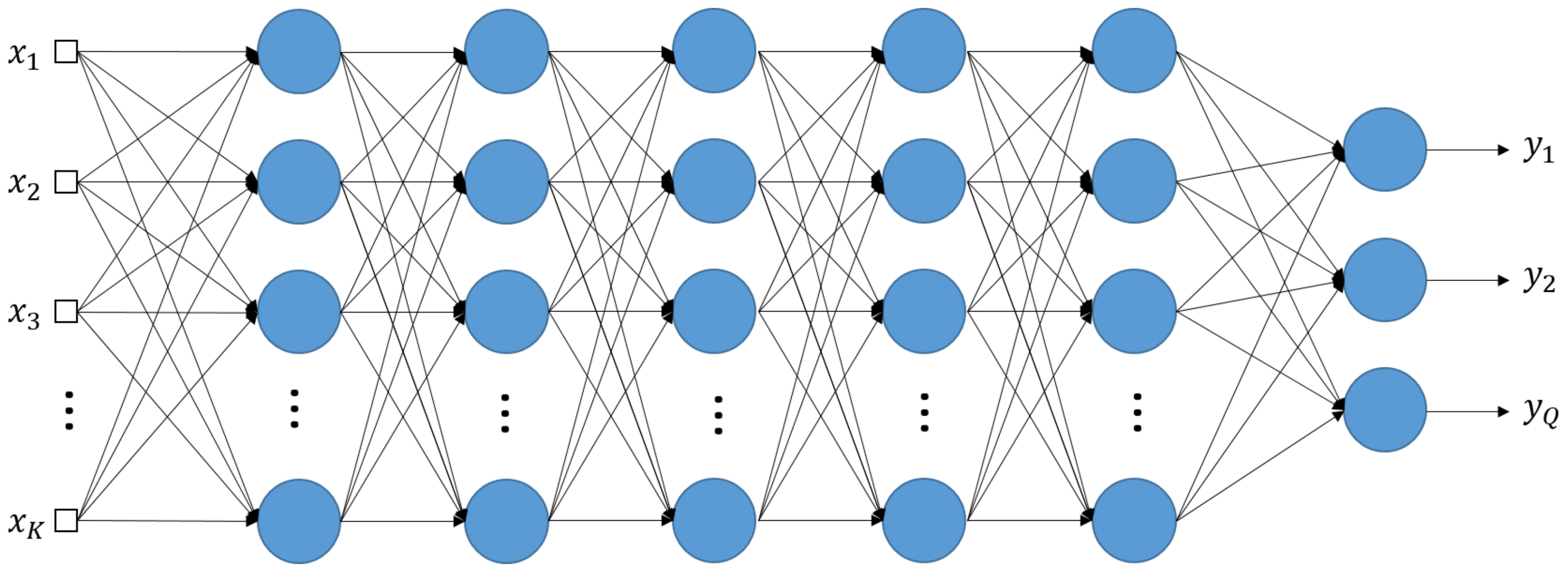
O problema da dissipação do gradiente

- Ocorre devido à natureza do ***algoritmo de retropropagação***, que é usado para treinar a rede neural.
  - Para atualizar os pesos de nós das camadas ocultas, calcula-se a derivada do erro de saída em relação àquele peso e, para isso, usamos a ***regra da cadeia***.
  - Ou seja, o algoritmo ***propaga o erro de saída para as camadas ocultas*** usando a ***regra da cadeia***.



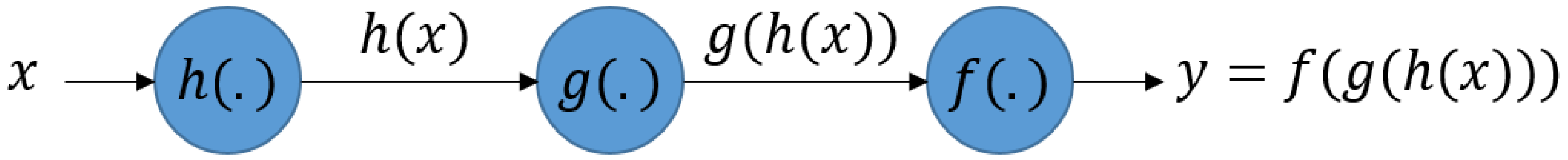
O problema da dissipação do gradiente

- Em suma, o ***vetor gradiente se torna cada vez menor*** conforme ele é calculado para as camadas próximas à entrada da rede, levando a uma ***atualização muito pequena ou até inexistente*** dos pesos destas camadas.



Regra da cadeia

- Durante o treinamento, para ***atualizar os pesos dos nós de cada camada*** da rede, o ***algoritmo de retropropagação*** calcula os vetores gradiente em relação aos pesos dessas camadas através da ***regra da cadeia***.
- Vejamos o exemplo abaixo com 3 nós e pesos das ligações iguais a 1.
  - **OBS**.: As funções $f(.), g(.), e\ h(.)$ podem ser interpretadas como sendo as funções de ativação dos nós.



- Como calculamos a derivada de y em relação à x?

$$\frac{\partial y}{\partial x} = \frac{\partial f\big(g(h(x))\big)}{\partial x} = \frac{\partial f\big(g(h(x))\big)}{\partial g(h(x))} \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial x}.$$

Regra da cadeia

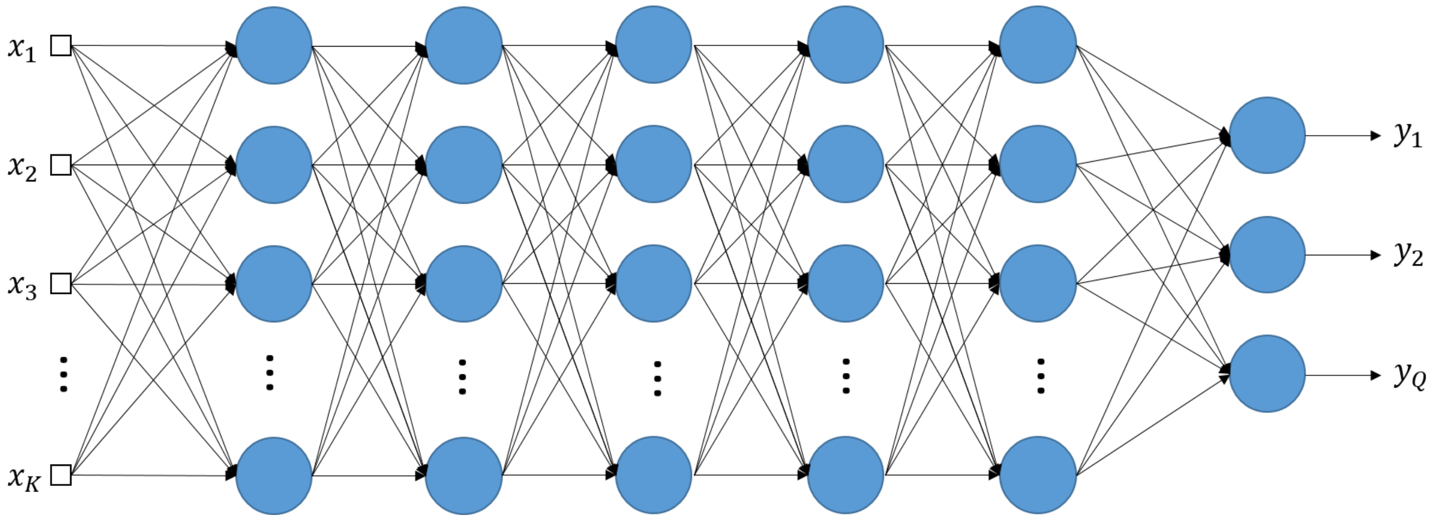
- Em outras palavras, devido à regra da cadeia, o ***vetor gradiente*** para a ***atualização dos pesos de uma dada camada*** da rede inclui o ***produto das derivadas das funções de ativação dos nós desde a camada de saída até a camada desejada***.

$$\frac{\partial y}{\partial x} = \frac{\partial f\big(g(h(x))\big)}{\partial x} = \frac{\partial f\big(g(h(x))\big)}{\partial g(h(x))} \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial x}.$$

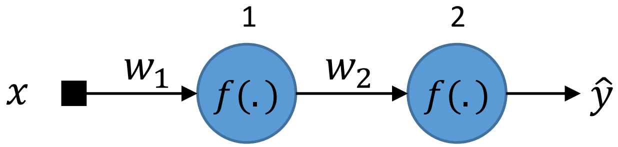
- Lembrem-se que as ***funções de ativação***, como ***tangente hiperbólica*** ou ***logística***, têm derivadas no intervalo de 0 até 1.
- Portanto, a multiplicação de vários termos menores do que 1 tende a 0 conforme o número de camadas da rede aumenta.

O problema da dissipação do gradiente

- Em uma rede com $\boldsymbol{M}$ camadas, a ***retropropagação*** tem o efeito de multiplicar até $\boldsymbol{M}$ valores pequenos (i.e., derivadas parciais das funções de ativação) para calcular os gradientes das primeiras camadas.
- O que significa que o ***gradiente diminui exponencialmente*** com $\boldsymbol{M}$.
- Isso significa que os ***nós das camadas iniciais aprendem muito mais lentamente do que os nós das camadas finais***, pois o valor do gradiente é muito pequeno, fazendo com que a ***atualização dos pesos também seja pequena*** (i.e., lenta).

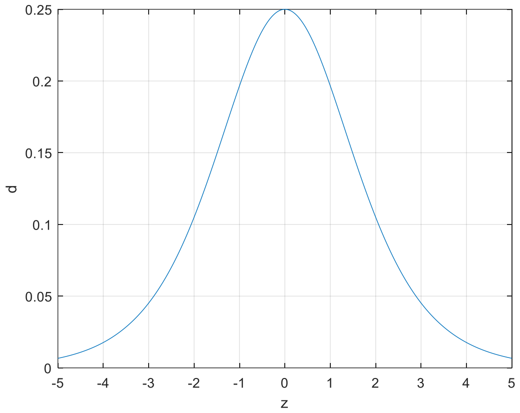


Exemplo: dissipação do gradiente



Considerações:

- 2 x Perceptrons com função de ativação sigmoide, $f(.)$.
- **Objetivo**: minimizar o erro quadrático médio, $J_e = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(i) - y(i))^2$.
- $g_1 = xw_1 \rightarrow$ entrada do primeiro perceptron.
- $a_1 = f(xw_1) \rightarrow$ saída do primeiro perceptron.
- $g_2 = a_1 w_2 = f(xw_1)w_2 \rightarrow$ entrada do segundo perceptron.
- $\hat{y} = f(f(xw_1)w_2) \rightarrow$ saída do segundo perceptron.
- As ***regras de atualização*** dos dois pesos são dadas por



$$w_2 = w_2 - \alpha \frac{\partial J_e}{\partial w_2}, \text{onde } \frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2},$$

$$w_1 = w_1 - \alpha \frac{\partial J_e}{\partial w_1}, \text{onde } \frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial a_1} \frac{\partial a_1}{\partial g_1} \frac{\partial g_1}{\partial w_1},$$

Derivadas da função de ativação.

onde $\frac{\partial J_e}{\partial w_1}$ e $\frac{\partial J_e}{\partial w_2}$ são obtidos com a ***regra da cadeia***.

Função de ativação retificadora

- Com o surgimento das ***redes neurais profundas***, uma outra função, conhecida como ***função retificadora***, passou a ser a bastante utilizada por questões ***computacionais e numéricas***.
- A ***função retificadora*** tem sua expressão dada por

$$y_j = f(z_j) = \max(0, z_j).$$

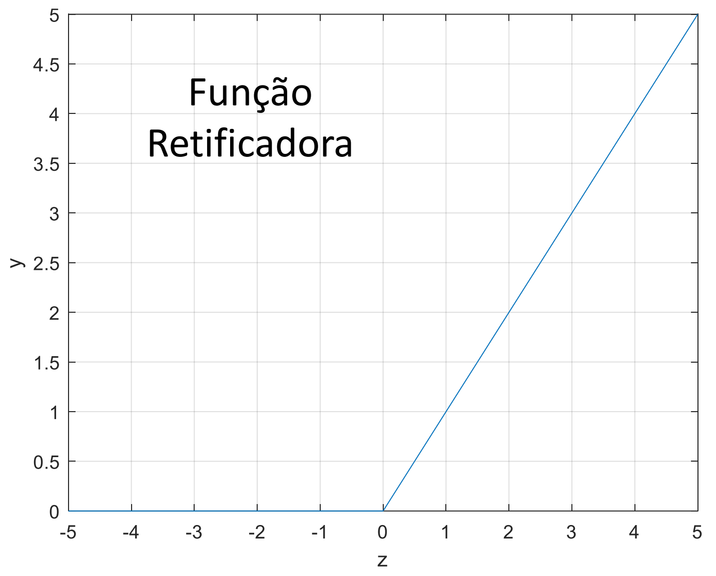
- Sua derivada é dada por

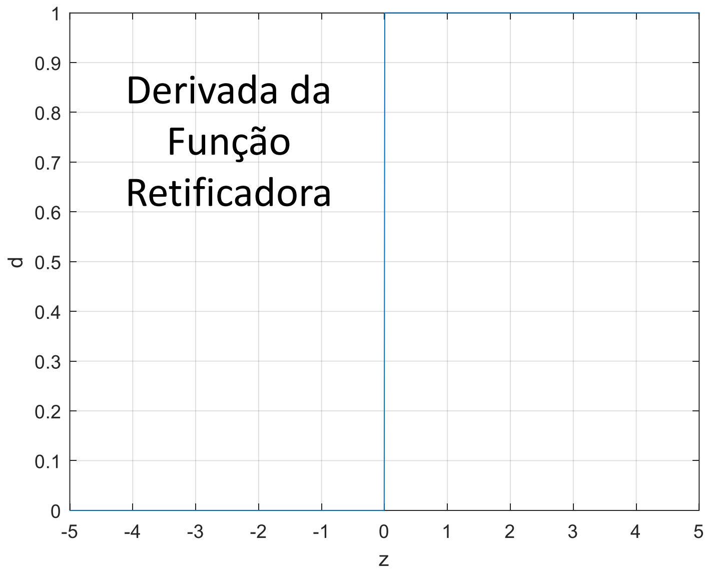
$$\frac{dy_j}{dz_j} = \frac{df(z_j)}{dz_j} = \begin{cases} 0, \text{se } z_j < 0 \\ 1, \text{se } z_j > 0' \end{cases}$$

Função degrau

e é indefinida para $z_j = 0$, porém o valor da derivada em zero pode ser arbitráriamente escolhido como 0 ou 1.

- Um ***nó*** que emprega uma ***função de ativação retificadora*** é chamado de ***rectified linear unit*** (ReLU)
- A ***função retificadora*** e sua derivada são mostradas nas figuras ao lado.





Função de ativação retificadora

- Vantagens da ***função retificadora***:
  - A função e sua derivada são ***mais rápidas de se calcular*** do que as funções logística e tangente hiperbólica.
  - Sofre menos com o ***problema da dissipação do gradiente***, pois sua derivada é igual 1 se $z_j > 0$. O produto da derivada da função de ativação ReLU dos nós de várias camadas sempre será igual a 1 se $z_j > 0$.
- Desvantagem
  - Entretanto, quando $z_j < 0$, o nó é considerado ***morto***, pois a derivada será igual a 0, fazendo com que os pesos permanecem inalterados (i.e., não há atualização).
- Outras funções de ativação são:
  - Parametric rectified linear unit (PReLU).
  - Leaky rectified linear unit (Leaky ReLU).

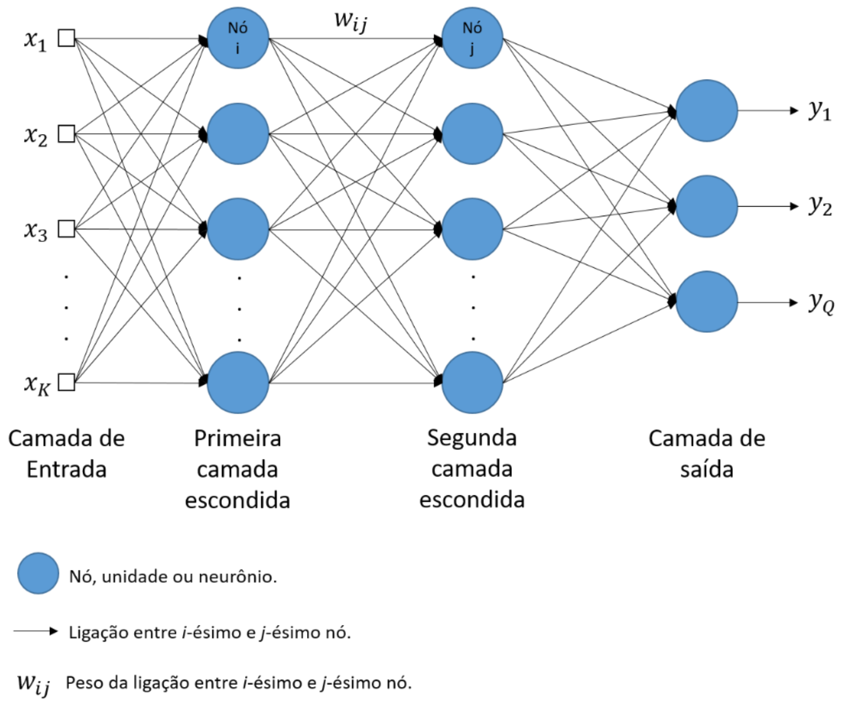
  Ambas têm derivada diferente de zero para $z_j < 0$.
  - https://en.wikipedia.org/wiki/Activation_function#Table_of_activation_functions
- Outras técnicas mais avançadas para evitar a dissipação do gradiente são a normalização de batch e o *dropout*.

Tarefa

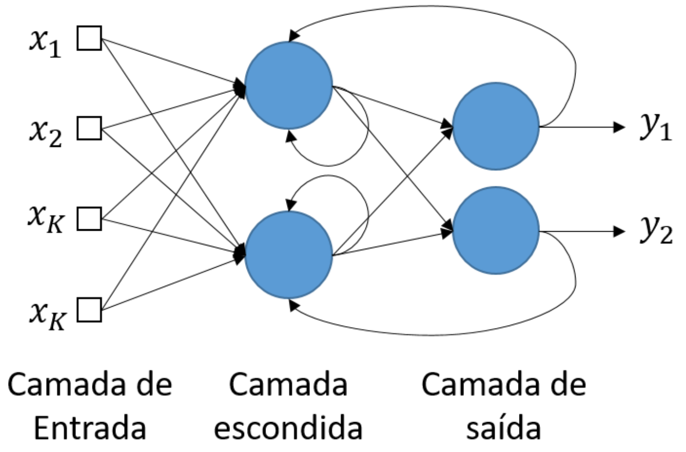
- **Quiz**: "*T320 - Quiz – Redes Neurais Artificiais (Parte III)*" que se encontra no MS Teams.

Conectando Neurônios

- Existem basicamente duas maneiras distintas para se conectar os ***nós*** de uma rede, ***direta*** e ***reversa***.
- Na figura ao lado, os ***nós*** da rede têm conexões em apenas uma única direção.
- Esse tipo de rede é conhecida como ***rede de alimentação direta*** (do inglês, *feedforward*) ou ***sem realimentação***.
- O sinal percorre a rede em uma única direção, da entrada para a saída.
- Os ***nós*** da mesma camada ***não são conectados entre si***.
- Esse tipo de rede representa uma ***função de suas entradas atuais*** e, portanto, não possui um estado interno além dos próprios pesos.

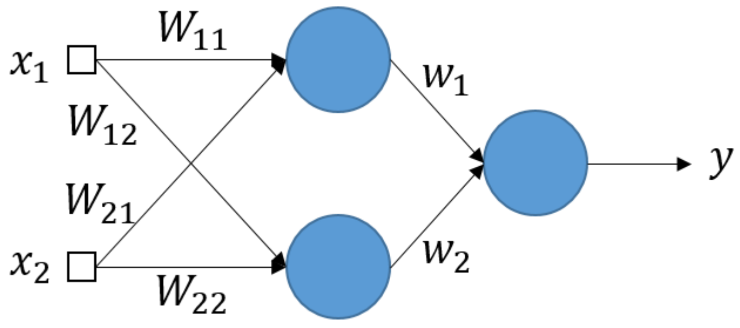


Conectando Neurônios



- Na figura acima, os ***nós*** da rede têm conexões em 2 direções, desta forma, o sinal percorre a rede nas direções ***direta e reversa***.
- Este tipo de rede é conhecida como ***rede recorrente*** ou ***rede com realimentação***.
- Nessas redes, a saída dos ***nós*** alimentam ***nós*** da mesma camada (inclusive o próprio ***nó***) ou de camadas anteriores.
- Isso significa que a rede forma um ***sistema dinâmico*** que pode atingir um estado estável, exibir oscilações ou mesmo um comportamento caótico, ou seja, divergir.
- Além disso, a saída da rede é ***função da entrada atual e de seu estado interno***, ou seja, de saídas anteriores.
- Portanto, ***redes recorrentes*** possuem ***memória***.
- Essas redes são úteis para o ***processamento de dados sequenciais***, como som, dados de séries temporais (preços de ações, padrões cerebrais, etc.) ou linguagem natural (escrita e fala).

Regressão Não-Linear



- A rede MLP da figura ao lado tem sua saída definida por

$$y = f\left(\boldsymbol{w}^T f(\boldsymbol{W}^T \boldsymbol{x})\right),$$

onde $f(.)$ é a ***função de ativação*** escolhida, $\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ e $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$.

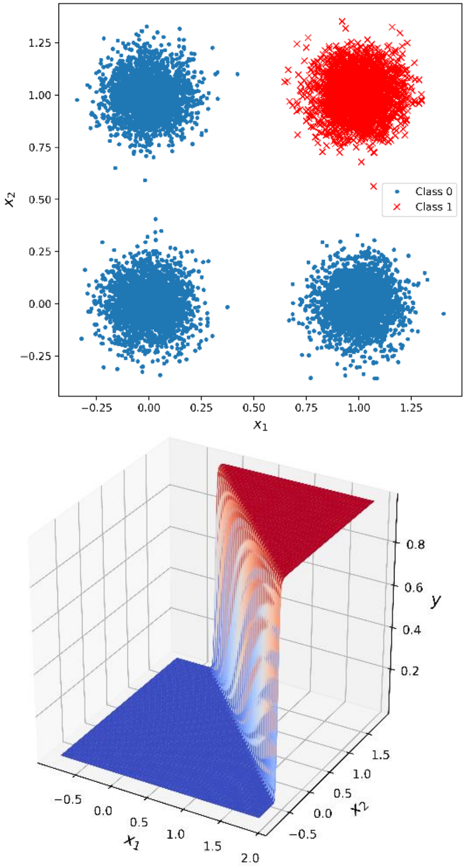
- Percebam que a saída da rede é dada pelo ***aninhamento*** das saídas de ***funções de ativação não-lineares***.
- Sendo assim, as funções que uma rede neural pode representar (i.e., aproximar) podem ser ***altamente não-lineares*** dependendo da quantidade de camadas e nós.
- Portanto, redes neurais podem ser vistas como ferramentas para a realização de ***regressão não-linear***, mas também podemos resolver outros problemas como os de classificação.
- Com ***uma única camada oculta suficientemente grande, é possível representar qualquer função contínua das entradas*** com uma precisão arbitrária (depende da topologia).
- Com ***duas camadas ocultas, até funções descontínuas podem ser representadas***.
- Portanto, dizemos que as redes neurais possuem ***capacidade de aproximação universal*** de funções.
- Veremos alguns exemplos desta capacidade de aproximação a seguir.

Aproximação universal de funções: Classificação

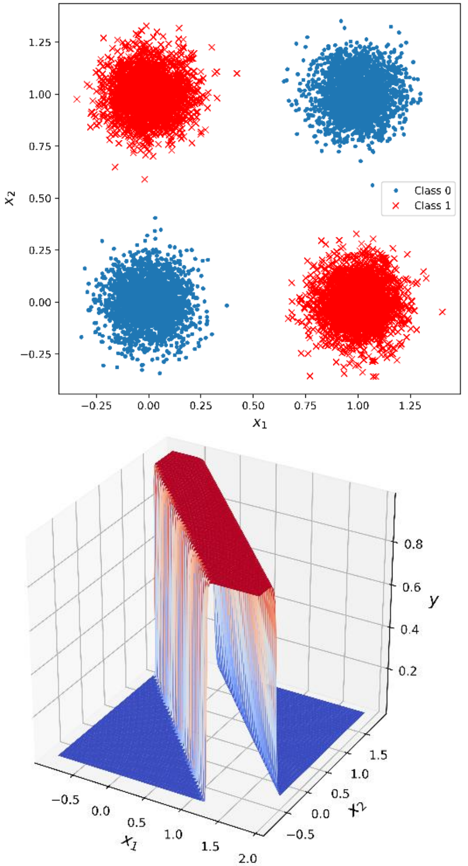
[Exemplo: FunctionApproximationWithMLP.ipynb]

- Fig. 1: Um nó aproxima uma função de limiar suave.
- Fig. 2: Combinando duas funções de limiar suave com direções opostas, podemos obter uma função em formato de onda.
- Fig. 3: Combinando duas ondas perpendiculares, nós obtemos uma função em formato cilíndrico.

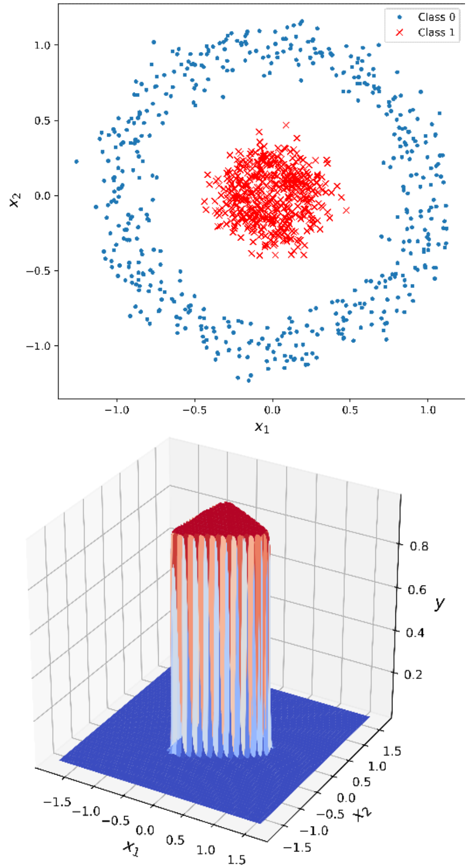
Função AND: MLP com 0 camadas escondidas, apenas um neurônio na camada de saída. Total: 1 nó.



Função XOR: MLP com 1 camada escondida com 2 nós. Total: 3 nós.



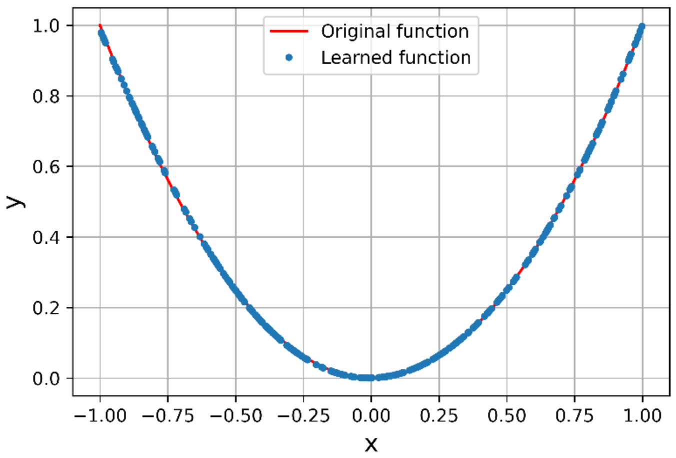
Círculos concêntricos: MLP com 1 camada escondida com 4 nós. Total: 5 nós.



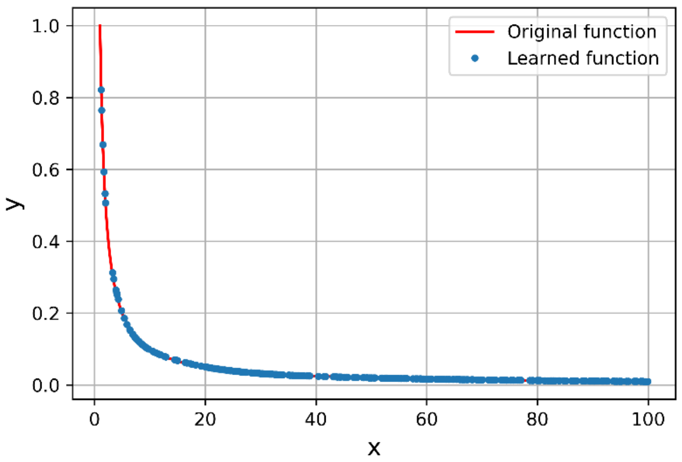
Aproximação universal de funções: Regressão

- Redes neurais podem ser usadas para aproximar funções como as mostradas abaixo:
    - $f(x) = x^2, -1 \leq x \leq 1,$
    - $f(x) = \frac{1}{x}, 1 \leq x \leq 100,$
    - $f(x) = \sin(x), 1 \leq x \leq 2\pi.$

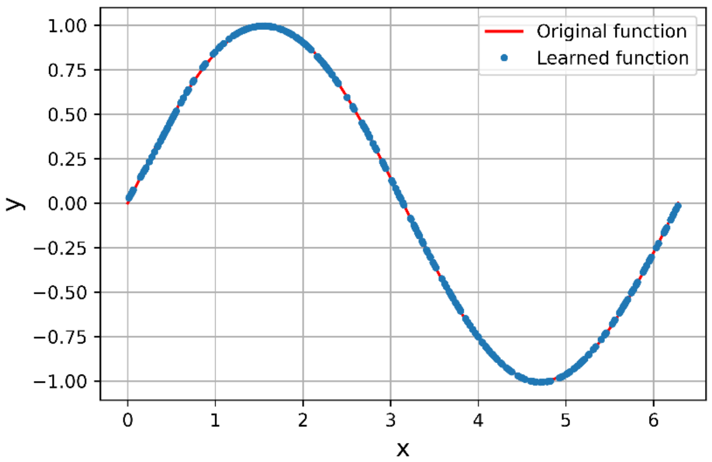
$f(x) = x^2$



$f(x) = \frac{1}{x}$



$f(x) = \sin(x)$



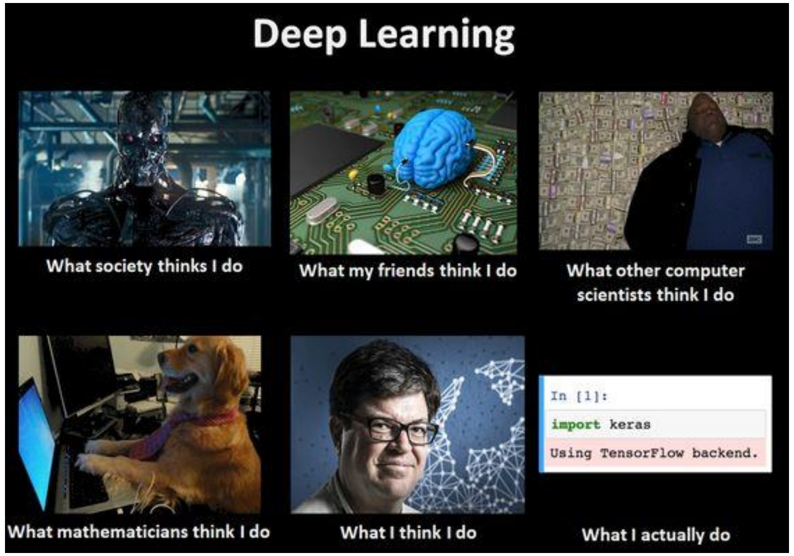
[Exemplo: function_approximation.ipynb]

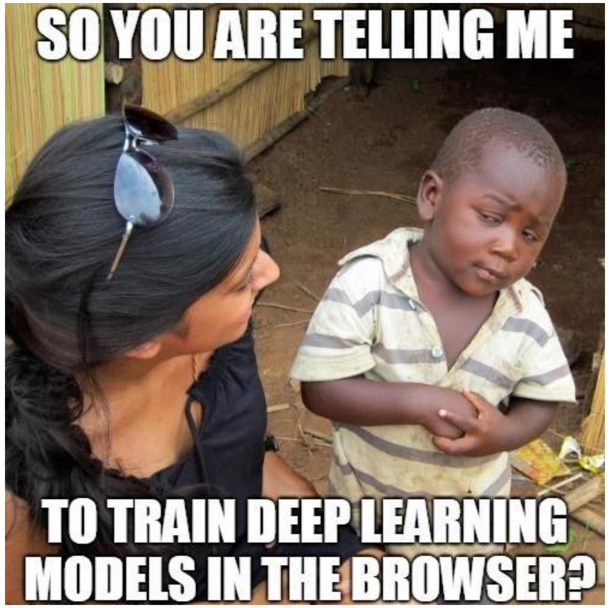
Tarefas

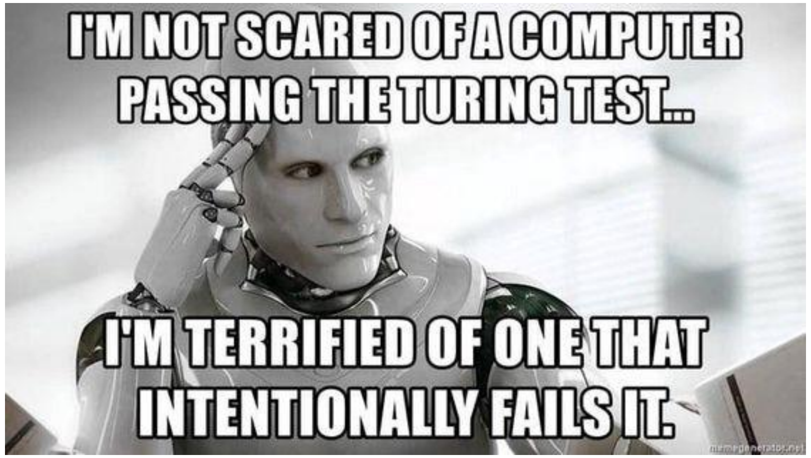
- **Quiz**: *“T320 - Quiz – Redes Neurais Artificiais (Parte IV)”* que se encontra no MS Teams.
- **Exercício Prático**: **Laboratório #7**.
  - Pode ser baixado do MS Teams ou do GitHub.
  - Pode ser respondido através do link acima (na nuvem) ou localmente.
  - Instruções para resolução e entrega dos laboratórios.
  - **Laboratórios podem ser feitos em grupo, mas as entregas devem ser individuais.**

Obrigado!


People with no idea about AI, telling me my AI will destroy the world
Me wondering why my neural network is classifying a cat as a dog..
Dog


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
In [1]:
import keras
Using TensorFlow backend.
What I actually do

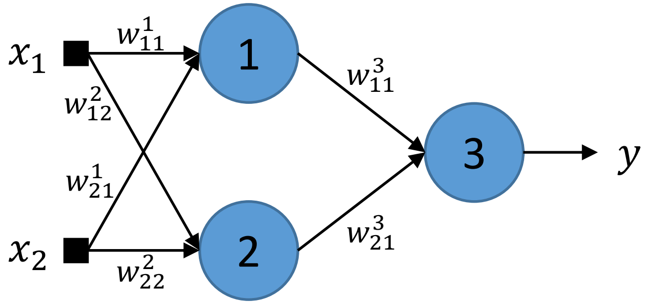

SO YOU ARE TELLING ME
TO TRAIN DEEP LEARNING MODELS IN THE BROWSER?

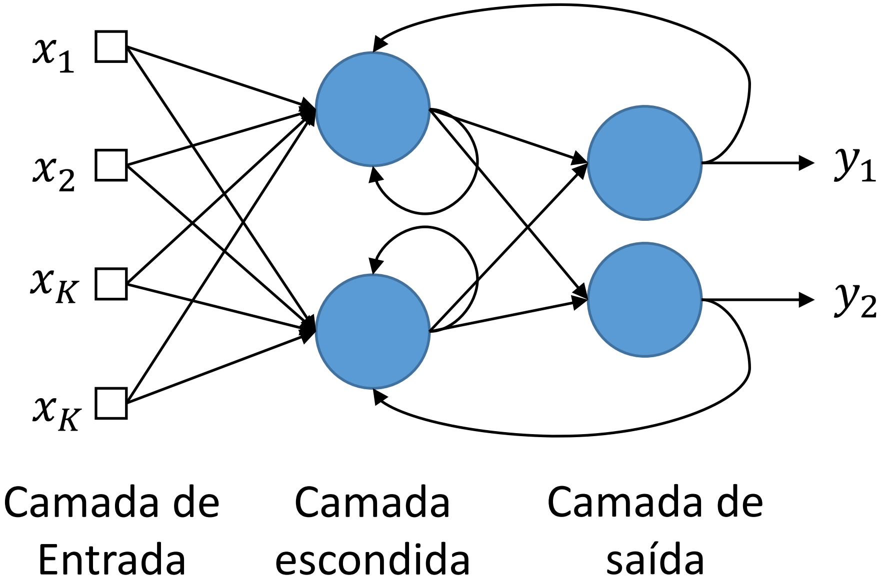

I'M NOT SCARED OF A COMPUTER PASSING THE TURING TEST...
I'M TERRIFIED OF ONE THAT INTENTIONALLY FAILS IT.


Dog
I NEED GPU
FOR MY DUMB
NEURAL NETWORK


ONE DOES NOT SIMPLY
GENERATE MEMES USING DEEP LEARNING

Figuras

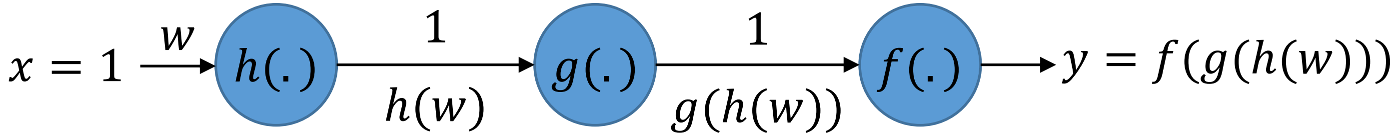

x_1
x_2
w_{11}^1
w_{12}^2
w_{21}^1
w_{22}^2
1
2
w_{11}^3
w_{21}^3
3
y

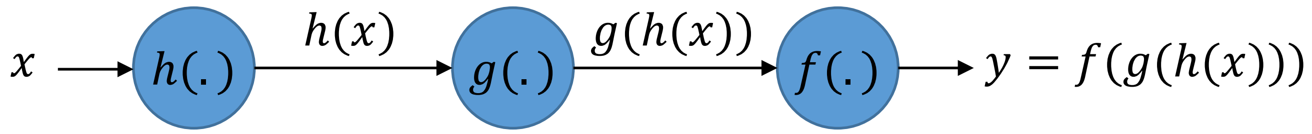

x_1
x_2
x_K
x_K
y_1
y_2
Camada de Entrada
Camada escondida
Camada de saída

O Problema da Dissipação do Gradiente

- Vamos entender esse problema através de um exemplo.
- Dada a simplificação de uma rede neural mostrada na figura abaixo, a qual contém
  - Três nós com as seguintes funções de ativação $h(.)$, $g(.)$ e $f(.)$.
  - Pesos w, 1 e 1, conectando os três nós, respectivamente.
  - Entrada $x = 1$.
- Para atualizarmos o valor do peso w com o gradiente descendente, precisamos encontrar a derivada parcial de y em relação à w.
- Para encontrar a derivada, usamos a regra da cadeia

$$\frac{\partial y}{\partial w} = \frac{\partial f\big(g(h(w))\big)}{\partial w} = \frac{\partial f\big(g(h(w))\big)}{\partial g(h(w))} \frac{\partial g(h(w))}{\partial h(w)} \frac{\partial h(w)}{\partial w}$$





x
$h(.)$
$h(x)$
$g(.)$
$g(h(x))$
$f(.)$
$y = f(g(h(x)))$