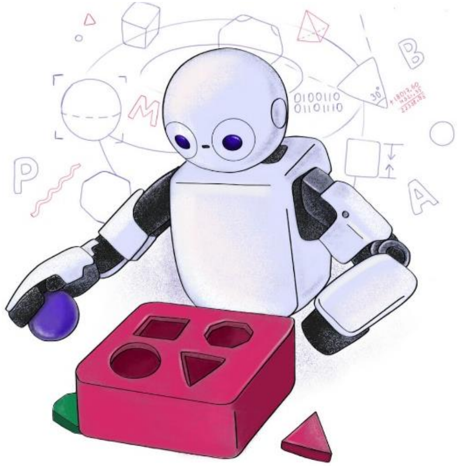
T320 - Introdução ao Aprendizado de Máquina II:

Redes Neurais Artificiais (Parte II)



Inatel

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

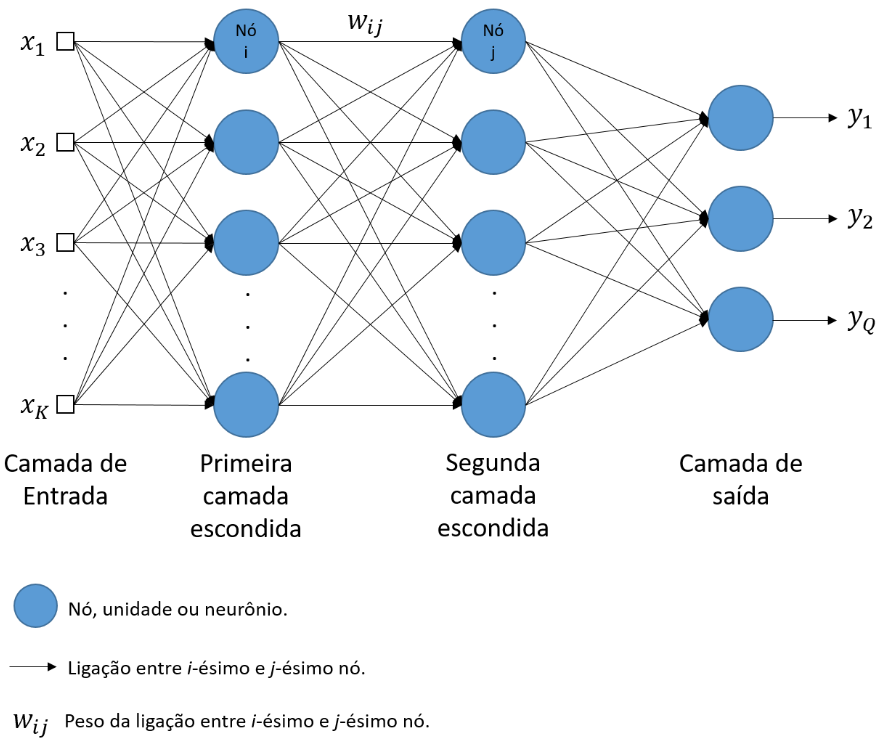
Recapitulando

- 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 do 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***.

Perceptron de Múltiplas Camadas

- Em termos gerais, uma ***rede neural*** nada mais é do que uma coleção de ***neurônios*** conectados entre si através de ***ligações direcionadas*** (ou seja, as conexões têm uma direção associada).
- As ***propriedades da rede neural*** são determinadas por sua ***topologia*** e pelas ***propriedades dos neurônios*** (e.g., função de ativação e pesos).
- Algumas das ***limitações dos perceptrons*** (e.g., classificação apenas de classes linearmente separáveis) podem ser ***eliminadas adicionando-se camadas intermediárias*** (também chamadas de ocultas ou escondidas) de ***perceptrons***.
- A RNA resultante é denominada ***Perceptron de Múltiplas Camadas*** (do inglês, *Multilayer Perceptron* - MLP).

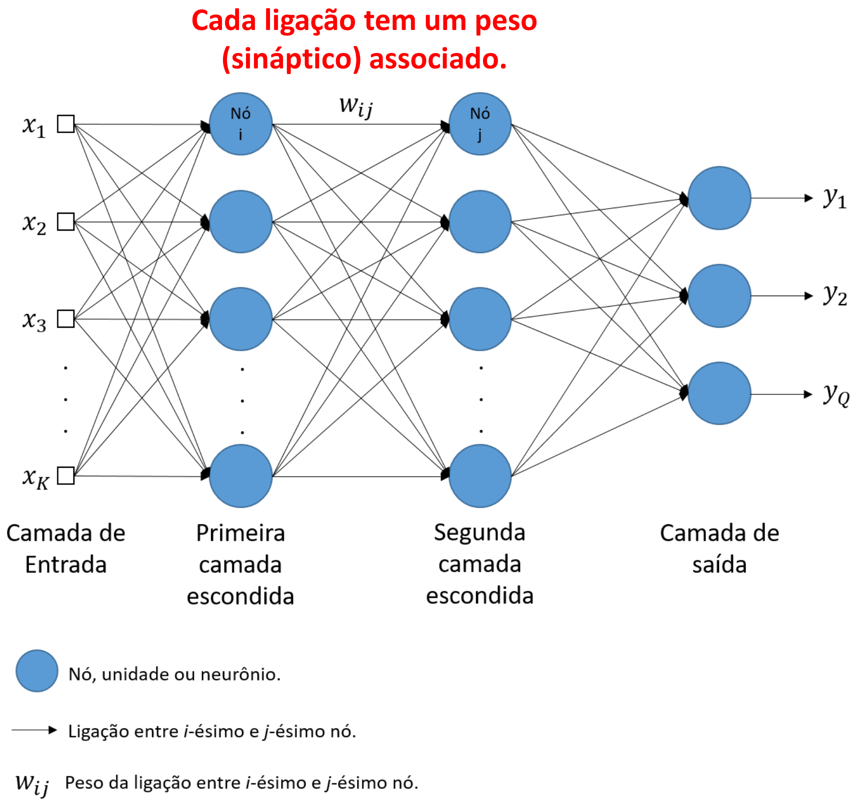
Cada ligação tem um peso (sináptico) associado.

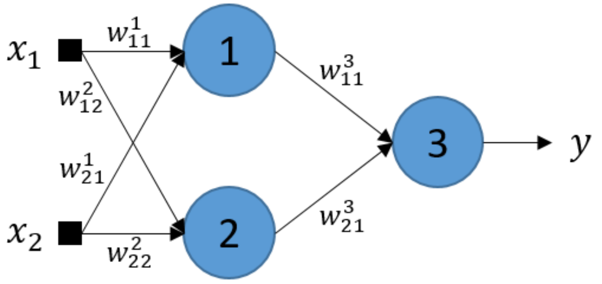


OBS.: Neurônios também são chamados de ***nós*** ou ***unidades***.

Perceptron de Múltiplas Camadas

- Uma rede MLP é sempre densamente conectada.
  - Cada nó em uma camada se conecta a cada nó na camada seguinte através de um peso sináptico.
- Um exemplo de rede ***MLP com duas camadas intermediárias*** é mostrado na figura ao lado.
- As RNAs são o coração do Deep Learning.
  - Quando uma RNA tem duas ou mais camadas escondidas, ela é chamada de ***rede neural profunda*** (ou em inglês *Deep Neural Network* - DNN).
- **OBS.**: Em particular, uma MLP pode resolver o problema do XOR.
  - Lembrem-se que um único ***perceptron*** não é capaz de realizar essa tarefa.





Perceptron de Múltiplas Camadas

- A ***camada de entrada*** é o ponto de transferência dos ***atributos*** à rede.
- As ***camadas intermediárias*** realizam ***mapeamentos não-lineares*** que, idealmente, vão tornando a informação contida nos dados mais ***"explícita"*** do ponto de vista da tarefa que se deseja realizar.
  - Os mapeamentos são ***não-lineares devido às funções de ativação*** utilizadas não serem lineares, e.g., função logística, tangente hiperbólica, etc.
- Por fim, os ***neurônios*** da ***camada de saída combinam a informação*** que lhes é ***oferecida pela última camada intermediária*** para formar as saídas.
- Redes MLPs são formadas por múltiplas camadas de ***Perceptrons***:
  - Portanto, tais redes têm por base o ***modelo de neurônio do Perceptron***.
- Esse modelo, discutido anteriormente, é mostrado na figura seguinte.

Perceptron de Múltiplas Camadas

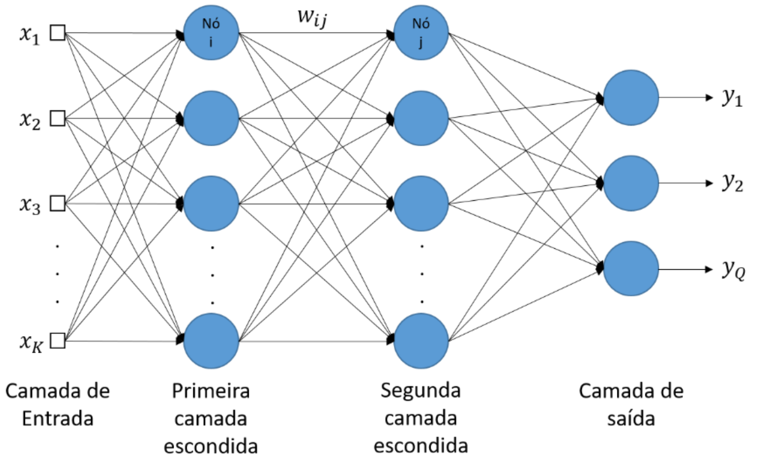
- A ***ligação*** do ***nó*** i para o ***nó*** j é feita através do ***peso*** w_{ij} e serve para ***propagar*** o sinal de ativação do ***nó*** i para o ***nó*** j.
- O valor do ***peso*** determina a ***força*** e o ***sinal*** da ***ligação***.
- Cada ***nó*** tem a entrada x_0 (o atributo de bias) sempre com valor igual a 1 e um peso associado w_{0j}.
  - Ou seja, esta entrada ***não está conectada a nenhum outro nó***.
- Cada ***nó*** j, calcula a ***soma ponderada*** de suas entrada da seguinte forma

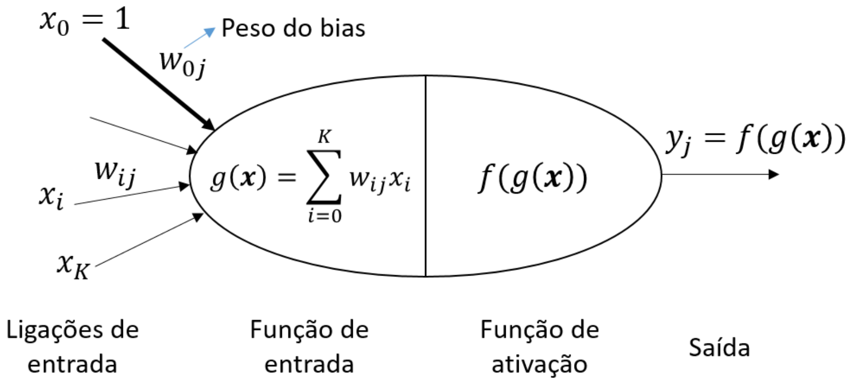
$$g(\boldsymbol{x}) = \sum_{i=0}^{K} w_{ij} x_i.$$

- Em seguida, o ***nó*** aplica uma ***função de ativação*** (i.e., de limiar), $f(.)$, ao somatório acima para obter sua saída

$$y_j = f(g(\boldsymbol{x})) = f\left(\sum_{i=0}^{N} w_{ij} x_i\right) = f(\boldsymbol{w}^T \boldsymbol{x}).$$

- Existem vários tipos de ***funções de ativação*** que podem ser utilizadas pelos ***nós*** de uma rede MLP.
- Cada camada pode usar funções de ativação diferentes, mas a mesma camada usa a mesma função, em geral.





$$y_j = f(g(\boldsymbol{x})) = f\left(\sum_{i=0}^{K} w_{ij} x_i\right),$$

onde x_i é a saída do nó i e w_{ij} é o peso conectando a saída do nó i para este nó, o nó j.

Funções de ativação

- Devido suas características, não se utiliza a ***função degrau*** como função de ativação em MLPs.
  - Derivada sempre igual a zero, exceto na origem, onde 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 ***possuem derivada definida e diferente de 0 em todos os pontos***.
- A ***função logística*** tem a seguinte expressão:

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

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

- Sua derivada é dada por

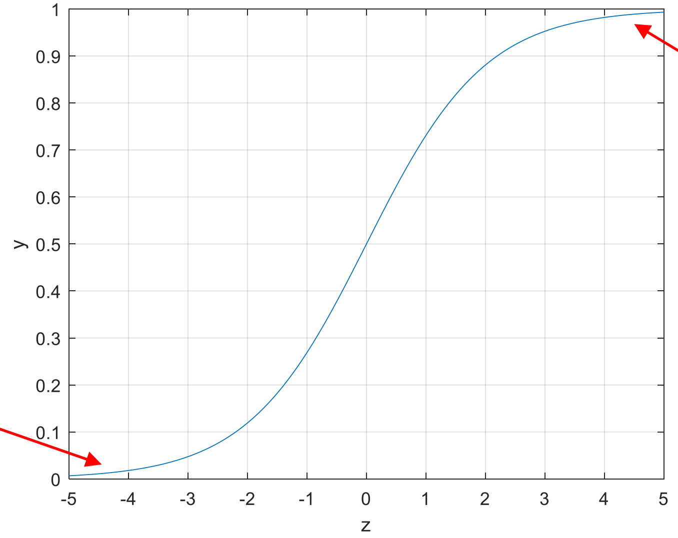
$$\frac{dy_j}{dz_j} = \frac{df(z_j)}{dz_j} = y_j(1 - y_j) \geq 0.$$

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

Funções de ativação

- A ***função logística*** e sua derivada são mostradas nas figuras abaixo.
- Percebam que o valor da derivada, d, ***sempre será menor do que 1, sendo no máximo igual a 0.25***.
- Na sequência, veremos que isso causa um problema no aprendizado de redes com muitas camadas, i.e., redes profundas.

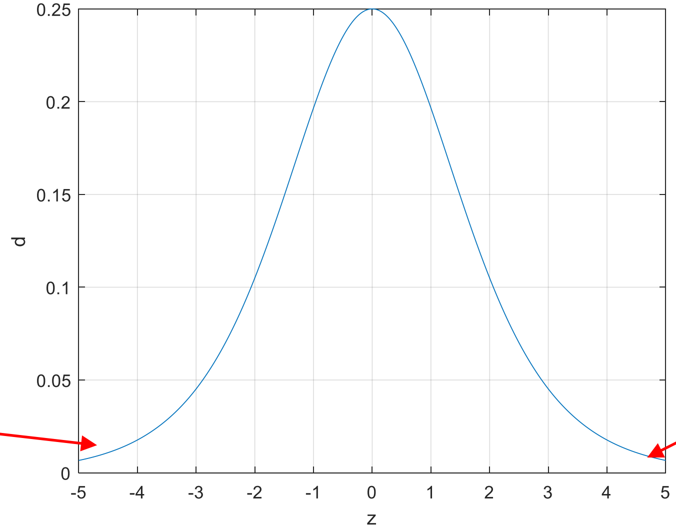
saturação:
valor de y tende a 0
quando $g(\boldsymbol{x}) \rightarrow -\infty$.



saturação:
valor de y tende a 1
quando $g(\boldsymbol{x}) \rightarrow \infty$.

Função Logística

saturação:
quando y tende a 0,
a derivada tende a 0.



saturação:
quando y tende a 1,
a derivada tende a 0.

Derivada da Função Logística

Funções de ativação

- A ***função tangente hiperbólica*** tem sua expressão dada por:

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

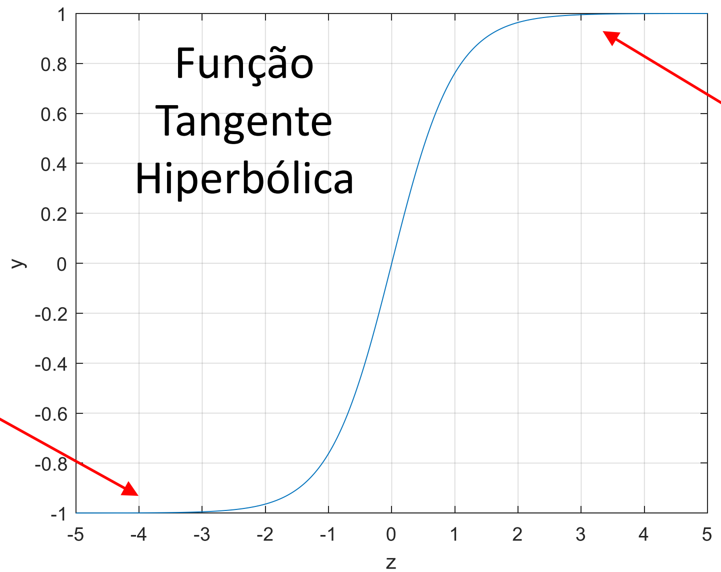
- Sua derivada é dada por

$$\frac{dy_j}{dz_j} = \frac{df(z_j)}{dz_j} = 1 - \tanh^2(z_j) > 0.$$

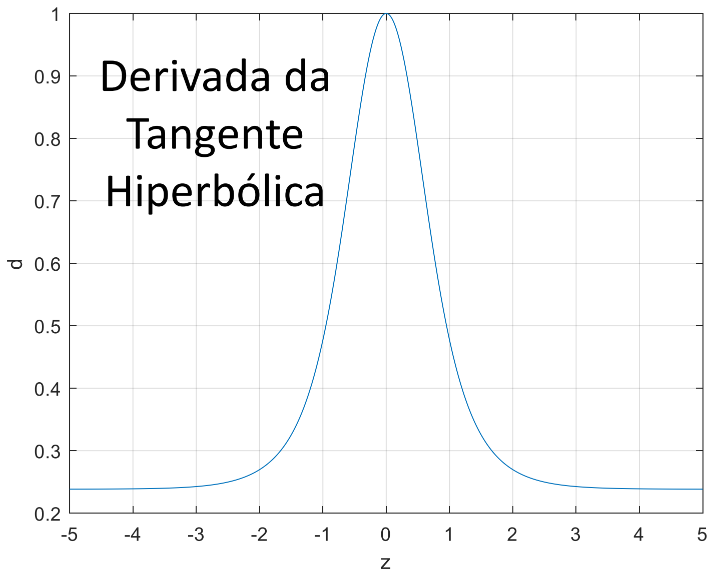
- A função e sua derivada são mostradas nas figuras abaixo.

A derivada é no máximo igual a 1 quando z, $g(\boldsymbol{x})$, é exatamente igual a 0.

saturação:
valor de y tende a 0 quando $g(\boldsymbol{x}) \to -\infty$.

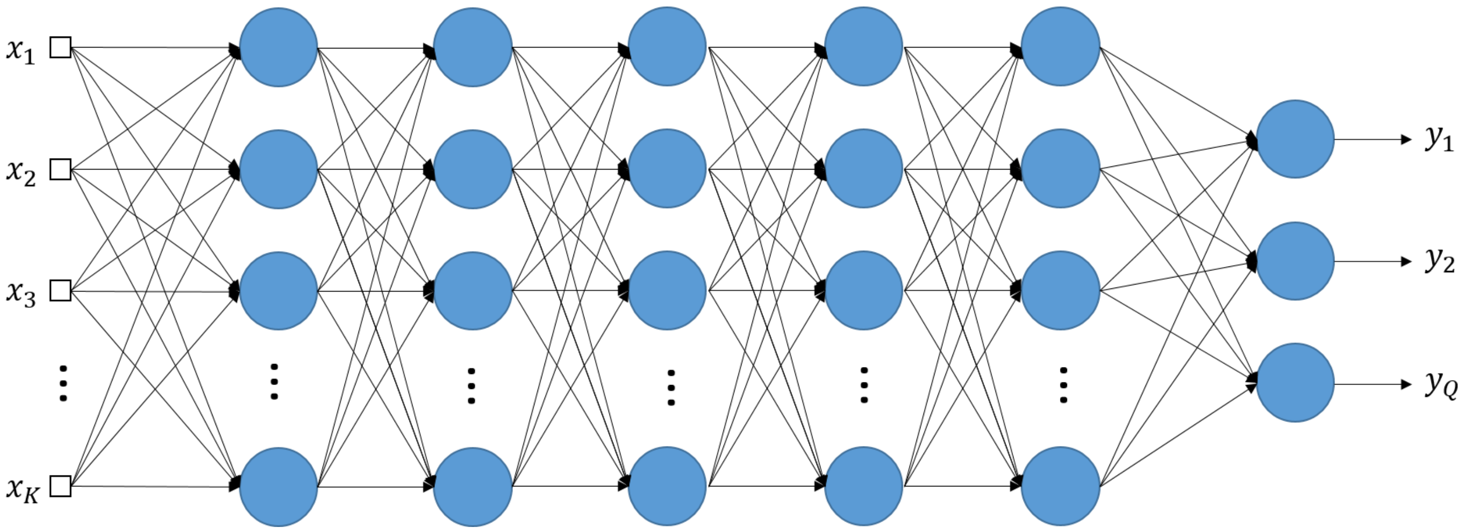


saturação:
valor de y tende a 1 quando $g(\boldsymbol{x}) \to \infty$.



O Problema da Dissipação do Gradiente

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

- Lembrem-se que as ***funções de ativação*** como ***tangente hiperbólica*** ou ***logística***, têm gradientes (i.e., derivadas parciais) no intervalo de 0 até 1.
- Durante o treinamento, para atualizar os pesos de cada camada da ***rede neural***, o ***algoritmo de retropropagação*** calcula os gradientes das camadas ocultas através do uso da ***regra da cadeia*** (exemplo abaixo).

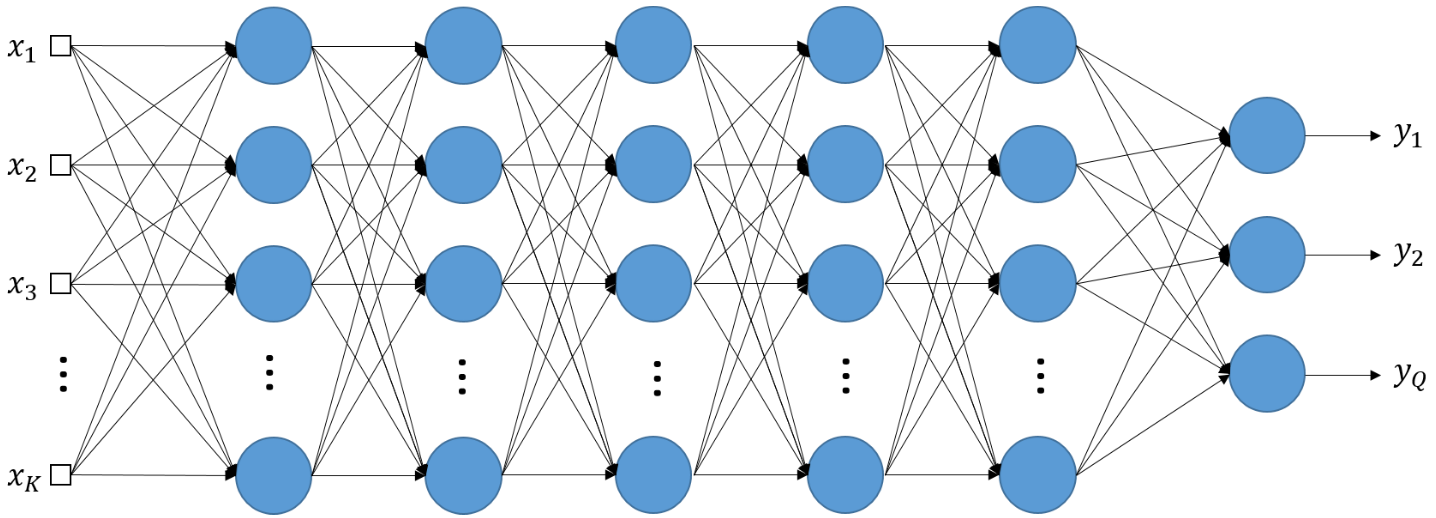
$$x \longrightarrow h(.) \xrightarrow{h(x)} g(.) \xrightarrow{g(h(x))} f(.) \longrightarrow y = f(g(h(x)))$$

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

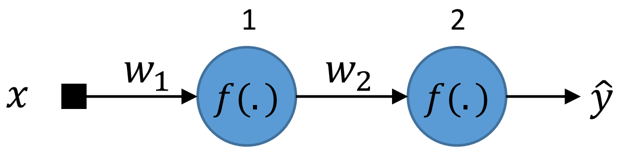
- Em outras palavras, devido à regra da cadeia, o gradiente para a atualização dos pesos de uma dada camada da rede neural inclui ***o produto das derivadas das funções de ativação desde a camada de saída até a camada desejada***.

O Problema da Dissipação do Gradiente

- Em uma rede com $\boldsymbol{M}$ camadas, a ***retropropagação*** tem o efeito de multiplicar $\boldsymbol{M}$ valores pequenos 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



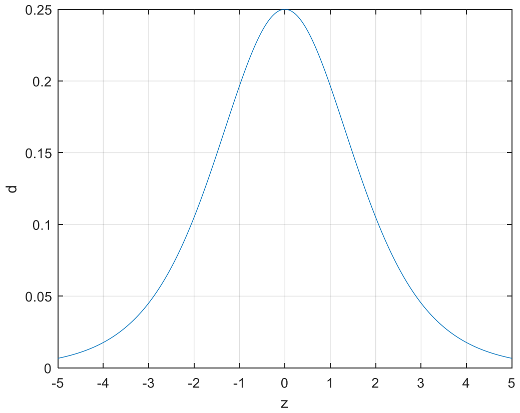
Considerações:

- 2 x Perceptrons com função de ativação sigmoide, $f(.)$.
- Minimização do 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

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

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

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



Derivadas da função de ativação.

Funções de ativação

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

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

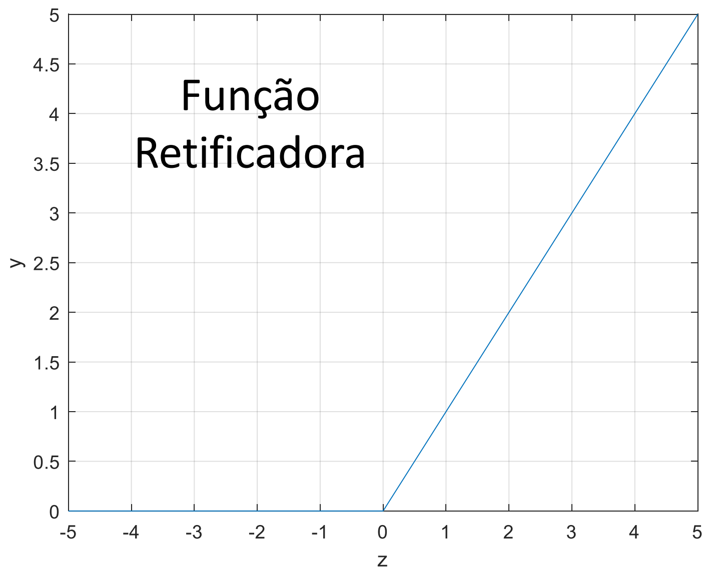
- Sua derivada é dada por

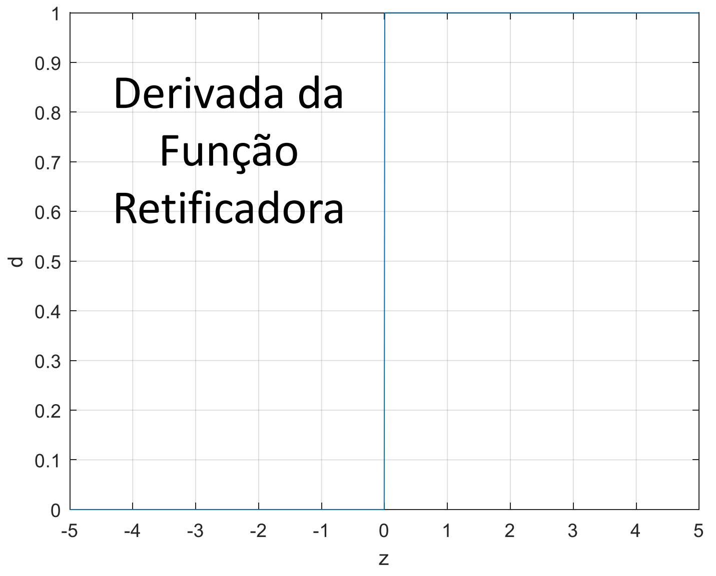
$$\frac{dy_j}{dz_j} = \frac{df\left(z_j\right)}{dz_j} = \begin{cases} 0, \text{se } z_j < 0 \\ 1, \text{se } z_j > 0' \end{cases}$$

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.





Funções de ativação

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

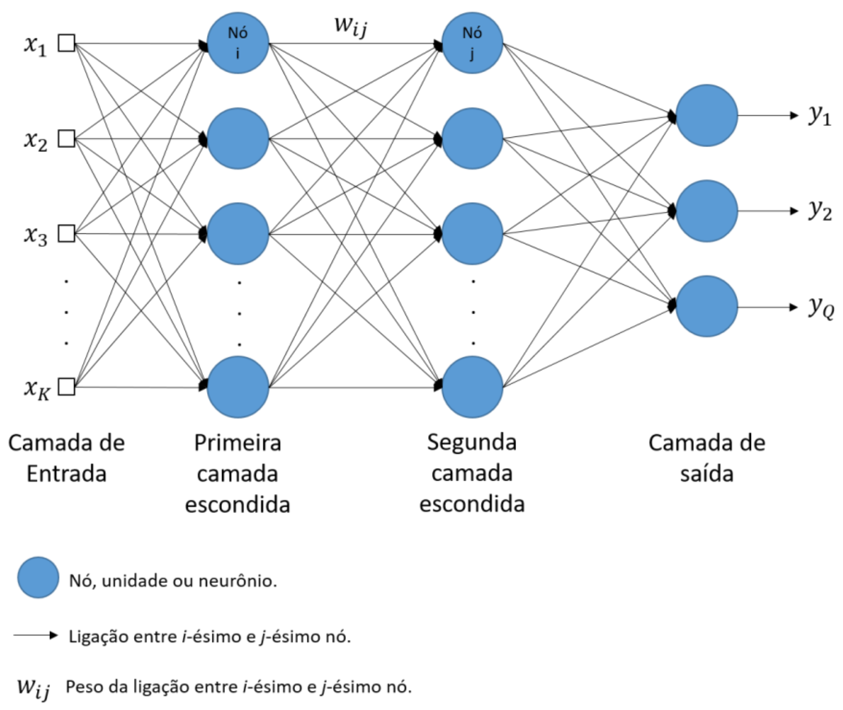
  Ambas têm gradiente diferente de zero para $z_j < 0$.
  - https://en.wikipedia.org/wiki/Activation_function#Table_of_activation_functions

Tarefa

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

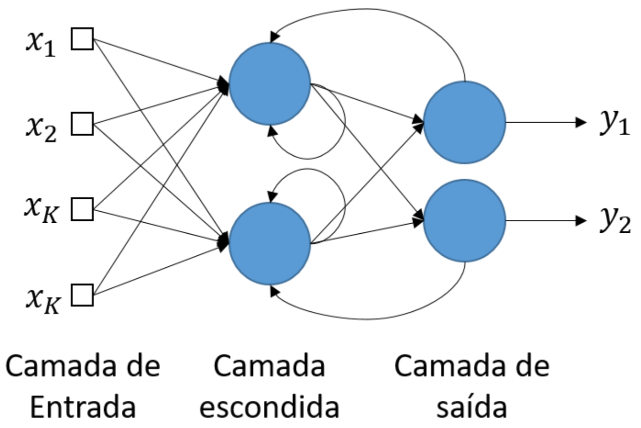
Conectando Neurônios

- 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.

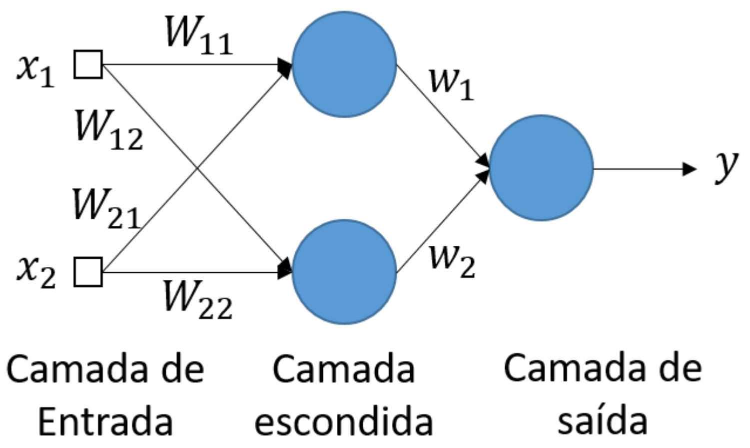


Conectando Neurônios

- Na figura ao lado, 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 entradas 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 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 $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$.

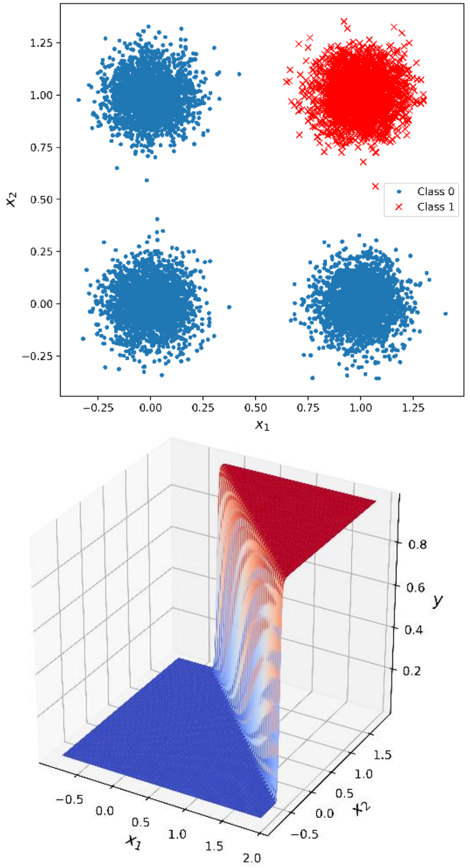
- Percebam que a saída da rede é dada pelo ***aninhamento*** das saídas de ***funções de ativação não-lineares***.
- Sendo assim, as funções que uma rede neural pode representar podem ser ***altamente não-lineares*** dependendo da quantidade de camadas e nós.
- Portanto, redes neurais podem ser vistas 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

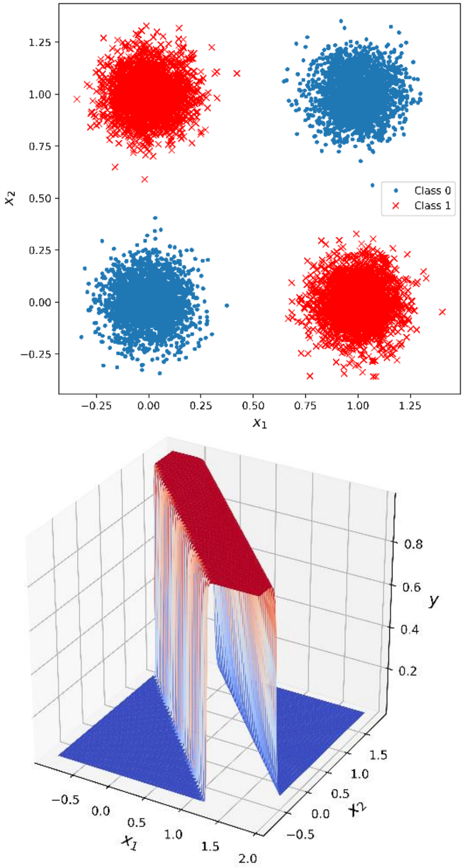
[Exemplo: FunctionApproximationWithMLP.ipynb](FunctionApproximationWithMLP.ipynb)

- 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.

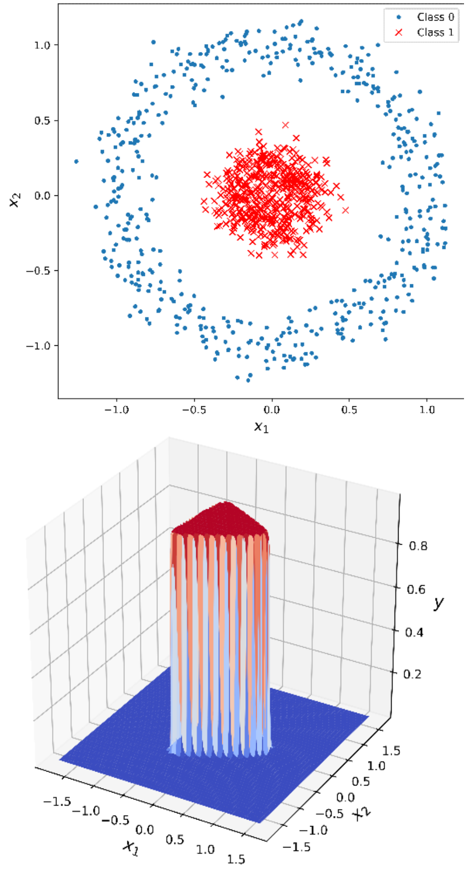
Função AND: MLP com 0 camadas escondidas, apenas um neurônio na camada de saída. Total: 1 nó.



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.

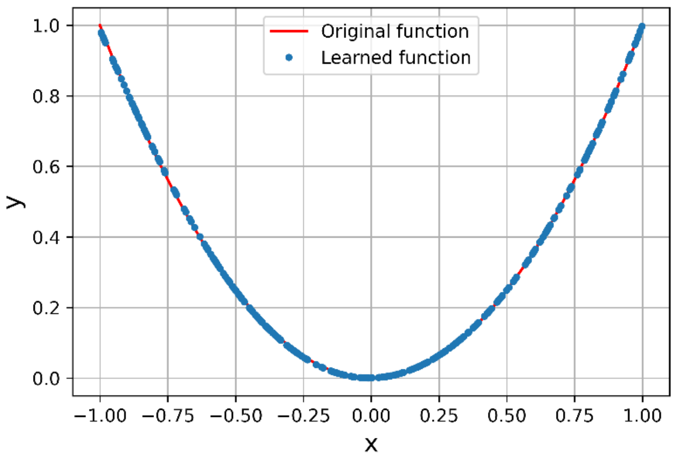


Aproximação universal de funções

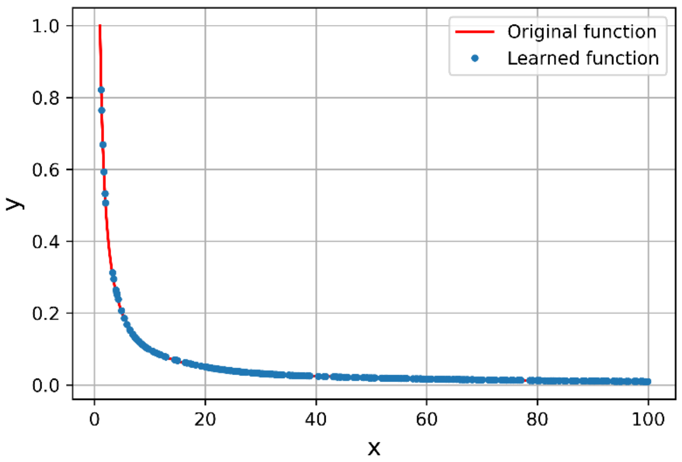
- Redes neurais podem ser usadas para aproximar funções como as mostradas abaixo:
  - $f(x) = x^2, -1 \leq x \leq 1,$
  - $f(x) = \frac{1}{x}, 1 \leq x \leq 100,$
  - $f(x) = \sin(x)\,, 1 \leq x \leq 2\pi.$

[Exemplo: function_approximation.ipynb]

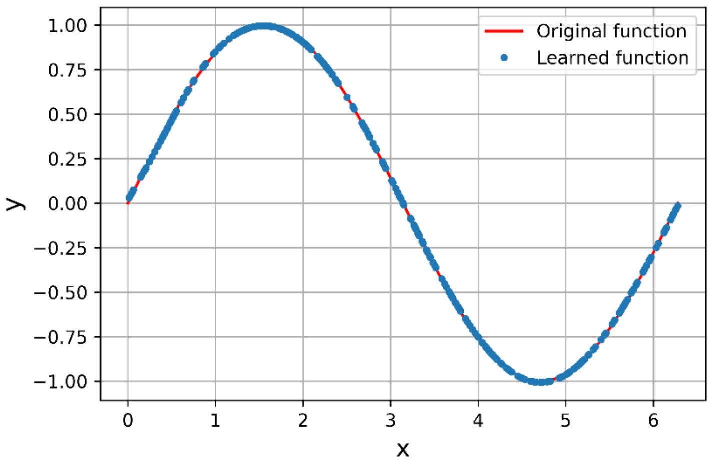
$f(x) = x^2$



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



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

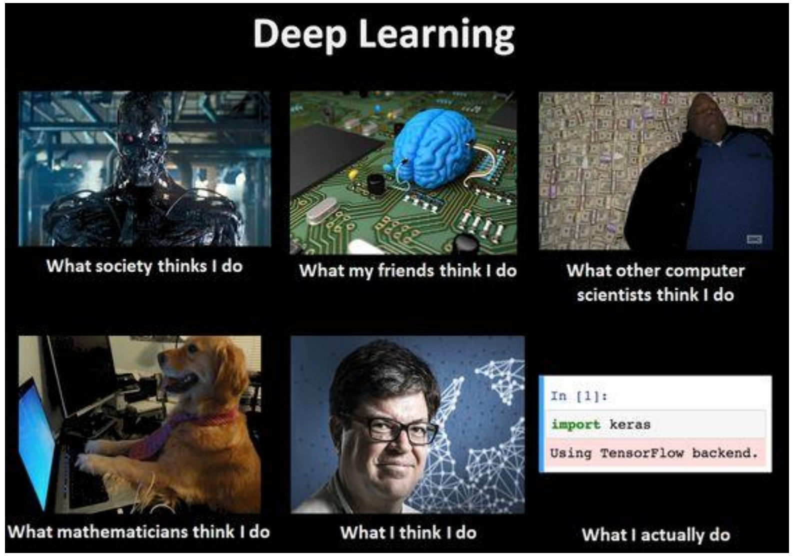


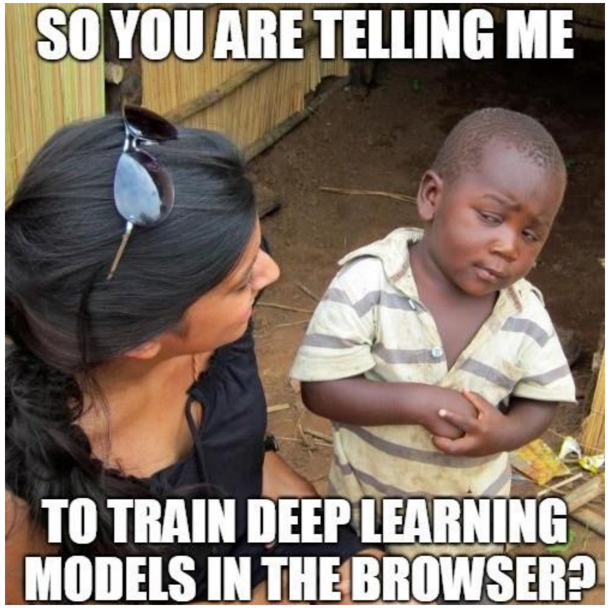
Tarefas

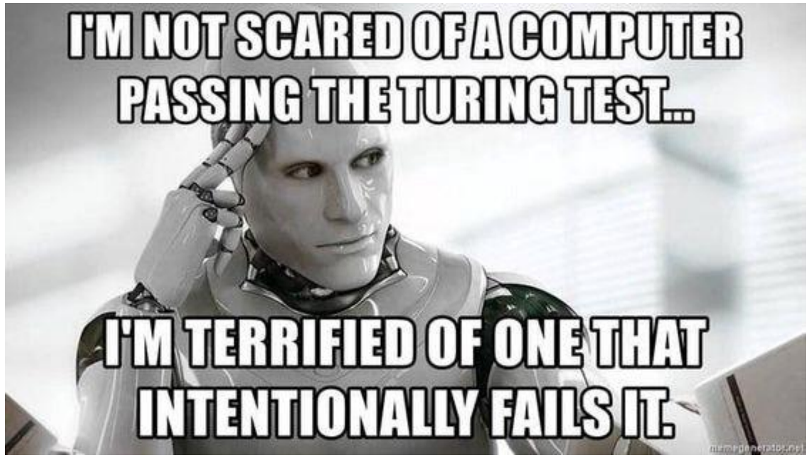
- **Quiz**: *"T320 - Quiz – Redes Neurais Artificiais (Parte IV)"* que se encontra no MS Teams.
- **Exercício Prático**: **Laboratório #7**.
  - Pode ser baixado do MS Teams ou do GitHub.
  - Pode ser respondido através do link acima (na nuvem) ou localmente.
  - Instruções para resolução e entrega dos 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

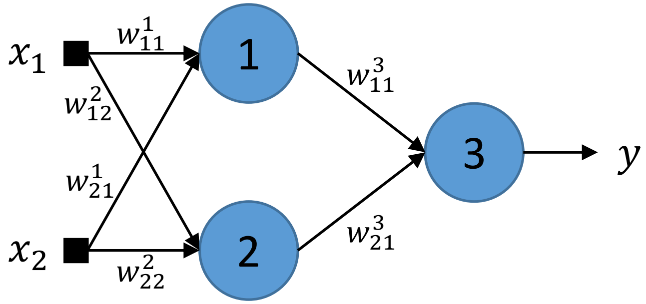

SO YOU ARE TELLING ME
TO TRAIN DEEP LEARNING MODELS IN THE BROWSER?


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

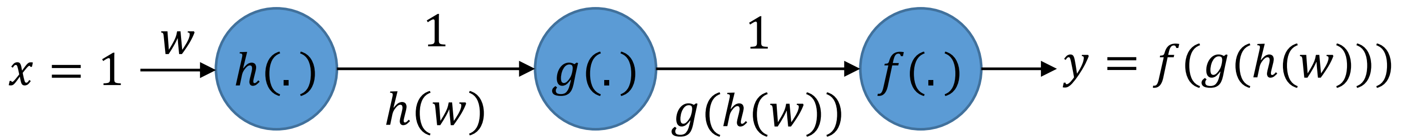
Figuras

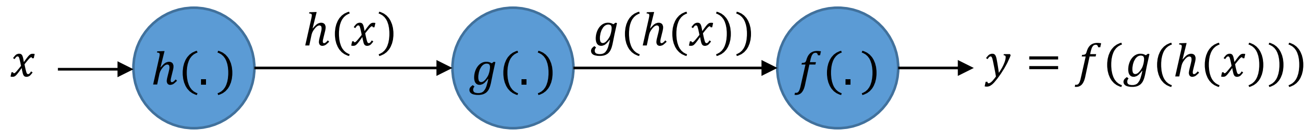

x_1
x_2
w_{11}^1
w_{12}^2
w_{21}^1
w_{22}^2
1
2
w_{11}^3
w_{21}^3
3
y

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