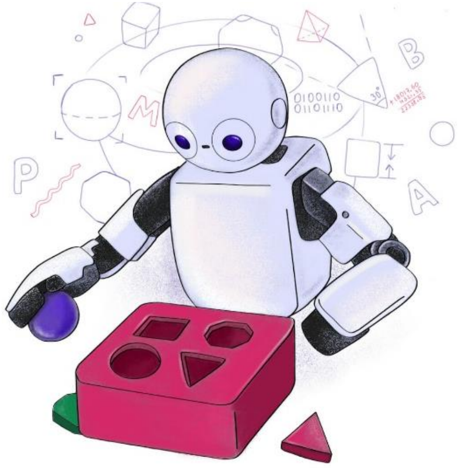
T320 - Introdução ao Aprendizado de Máquina II:

Redes Neurais Artificiais (Parte II)

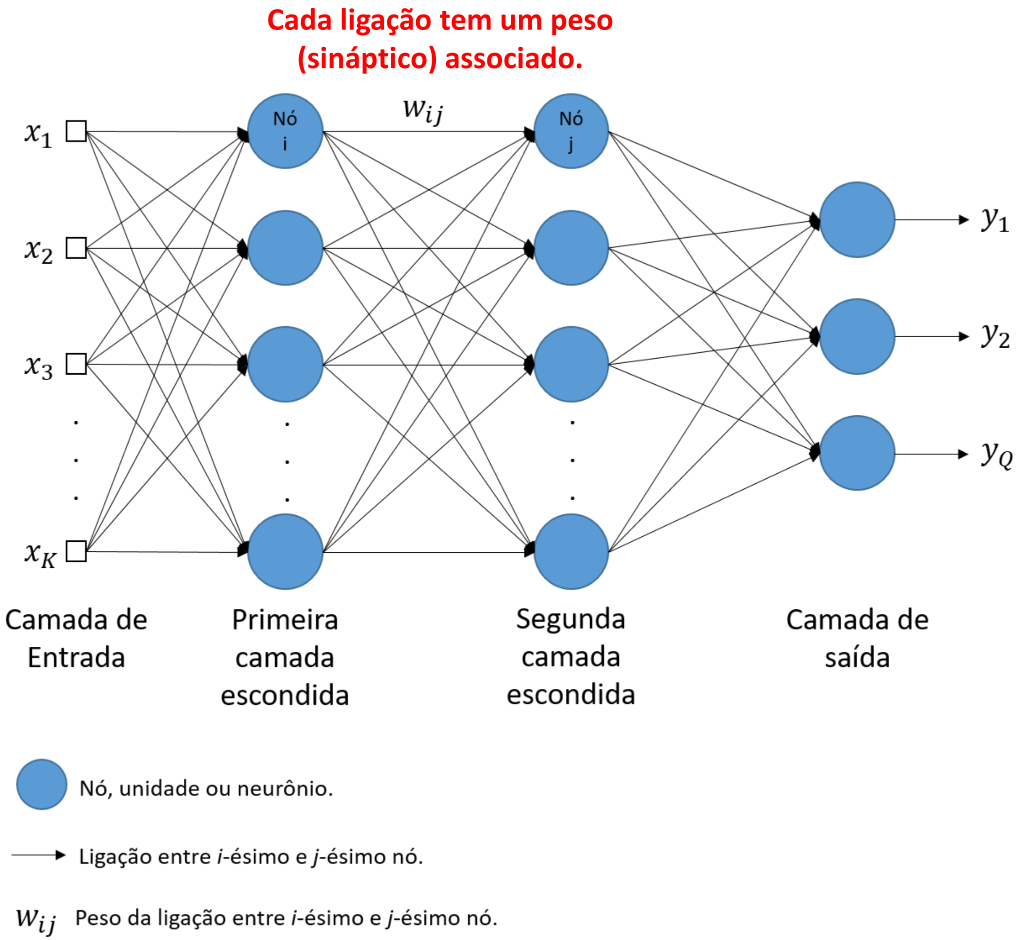


Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Recapitulando

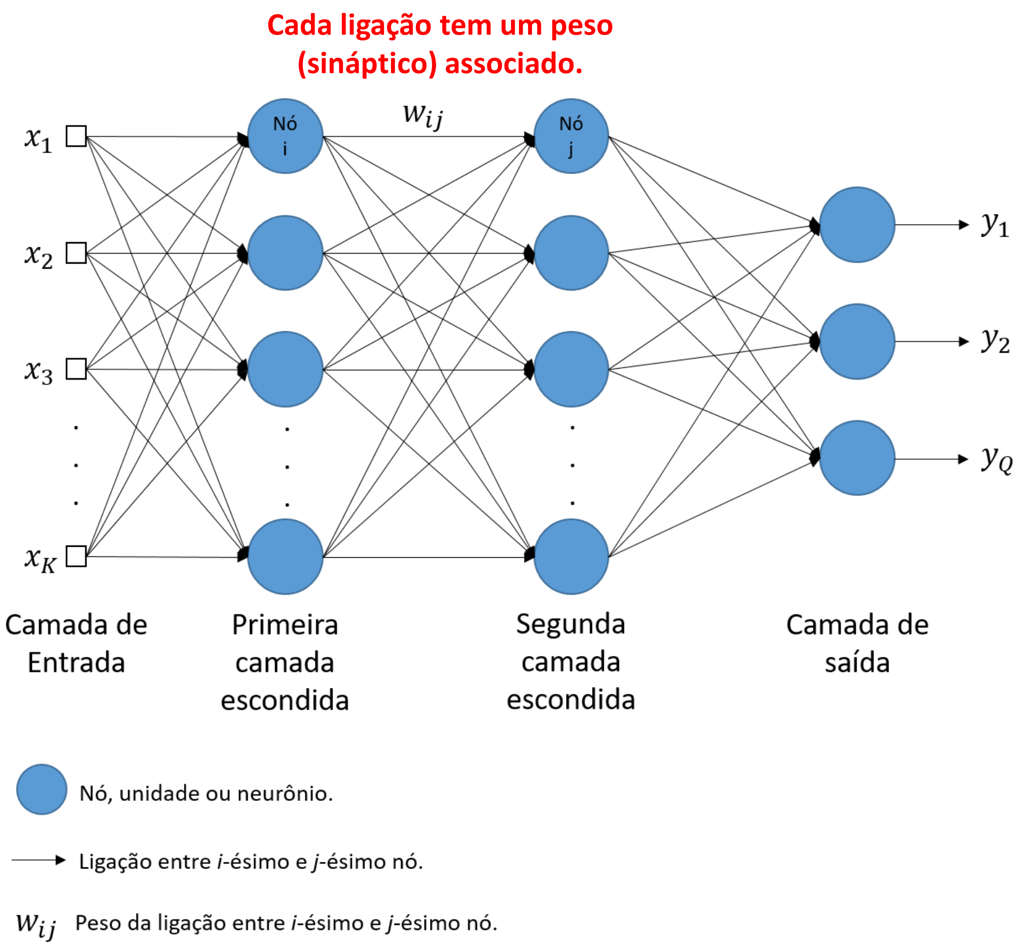
- Fizemos uma analogia entre um neurônio e os modelos de McCulloch e Pitts e do Perceptron.
- Vimos a evolução do modelo de McCulloch e Pitts para o Perceptron.
- Aprendemos suas características, diferenças e como ambos funcionam.
- Verificamos que um Perceptron é semelhante ao regressor logístico.
- 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



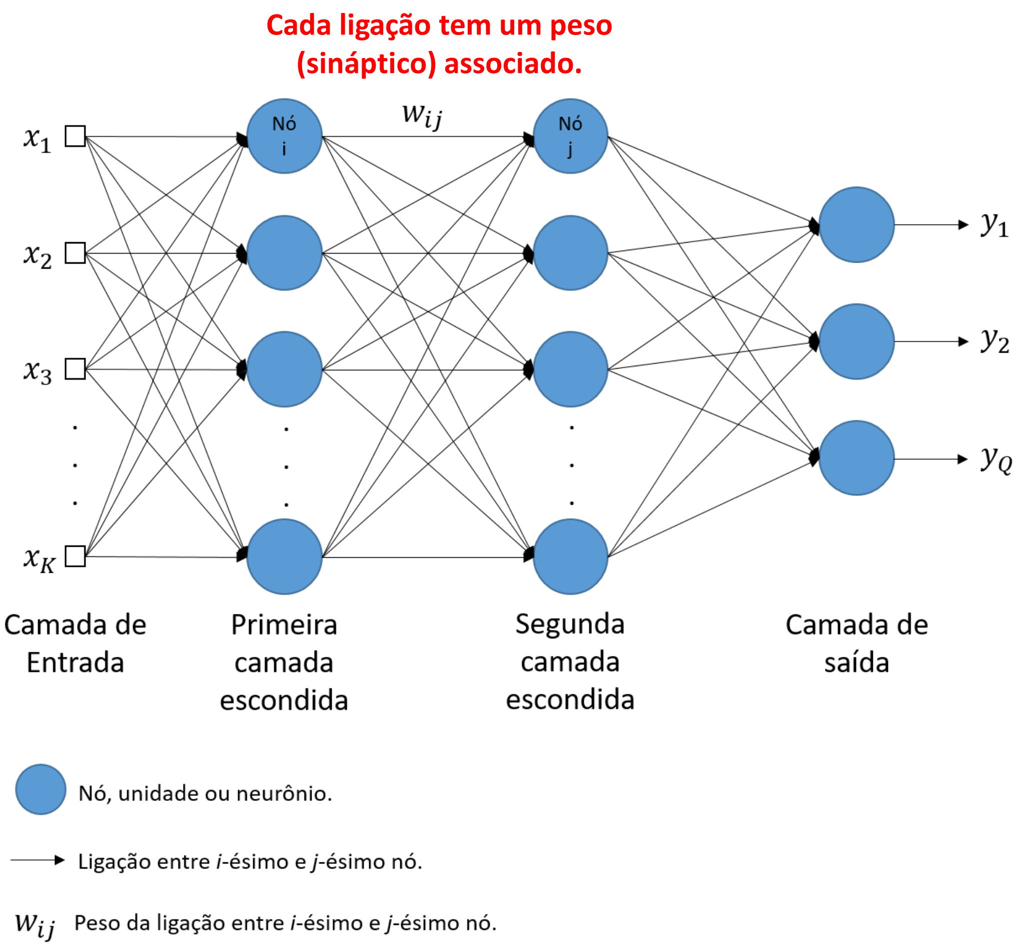
- Uma ***rede neural artificial (RNA)*** nada mais é do que uma ***combinação de neurônios*** conectados entre si através de ***ligações direcionadas*** (ou seja, as conexões têm uma direção associada).
  - Neurônios também são chamados de ***nós*** ou ***unidades***.
  - ***Cada ligação*** entre nós ***possui um peso (sináptico) associado***.

Perceptron de múltiplas camadas



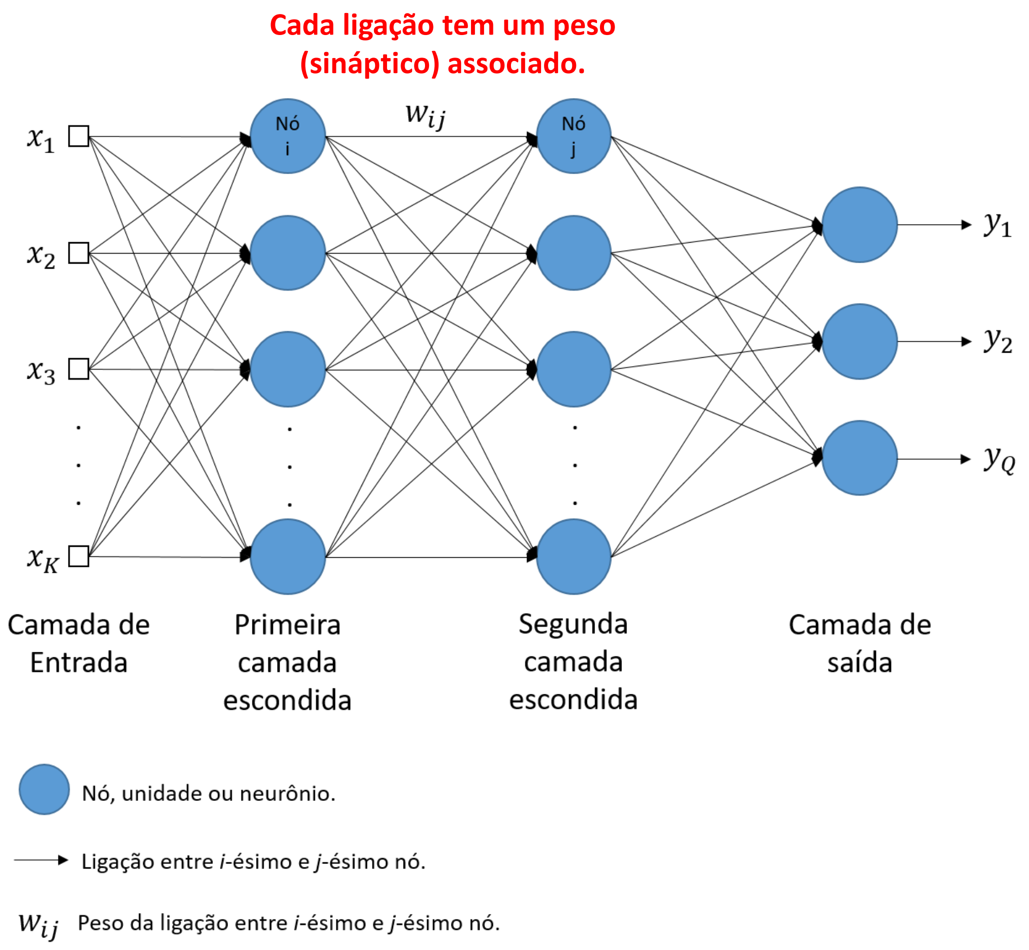
- As ***propriedades da rede neural*** são determinadas por sua ***arquitetura***, i.e., como os neurônios estão conectados, quantidade neurônios e de camadas escondidas, função de ativação, etc.

Perceptron de múltiplas camadas



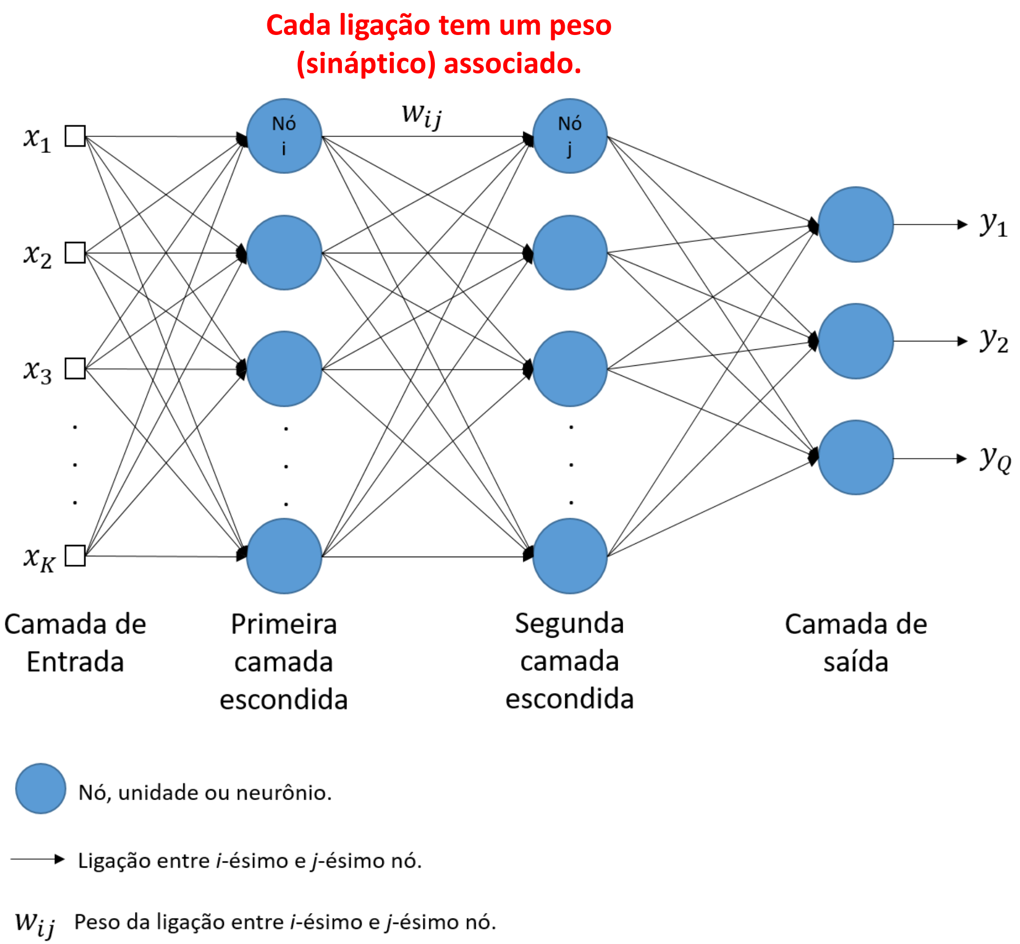
- Algumas das ***limitações dos perceptrons*** (e.g., classificação apenas de classes linearmente separáveis) podem ser ***superadas adicionando-se camadas intermediárias*** de ***perceptrons***.
- As camadas intermediárias são também chamadas de ***ocultas*** ou ***escondidas***.

Perceptron de múltiplas camadas



- A RNA 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.

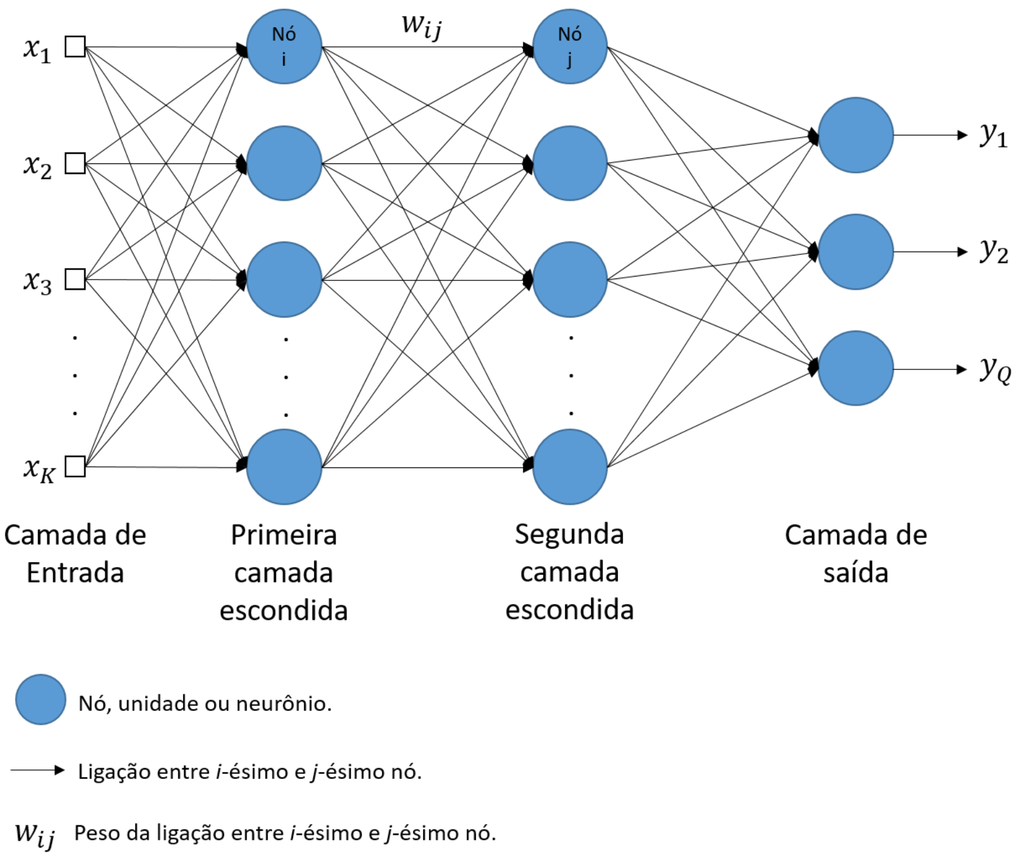
Perceptron de múltiplas camadas



- Essa rede é chamada de ***perceptron de múltiplas camadas*** (do inglês, *Multilayer Perceptron* - MLP) ou de ***rede densamente conectada (de alimentação direta)*** (do inglês, *Dense Neural Network* - DNN).

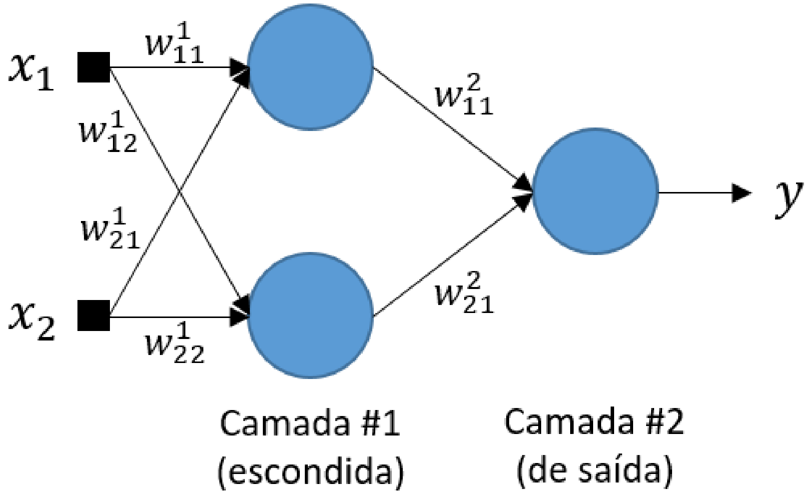
Perceptron de múltiplas camadas

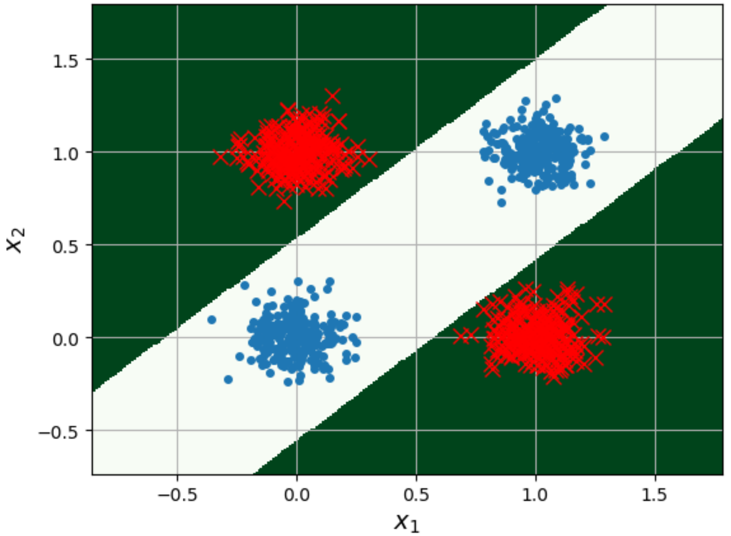
Cada ligação tem um peso (sináptico) associado.



- 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.

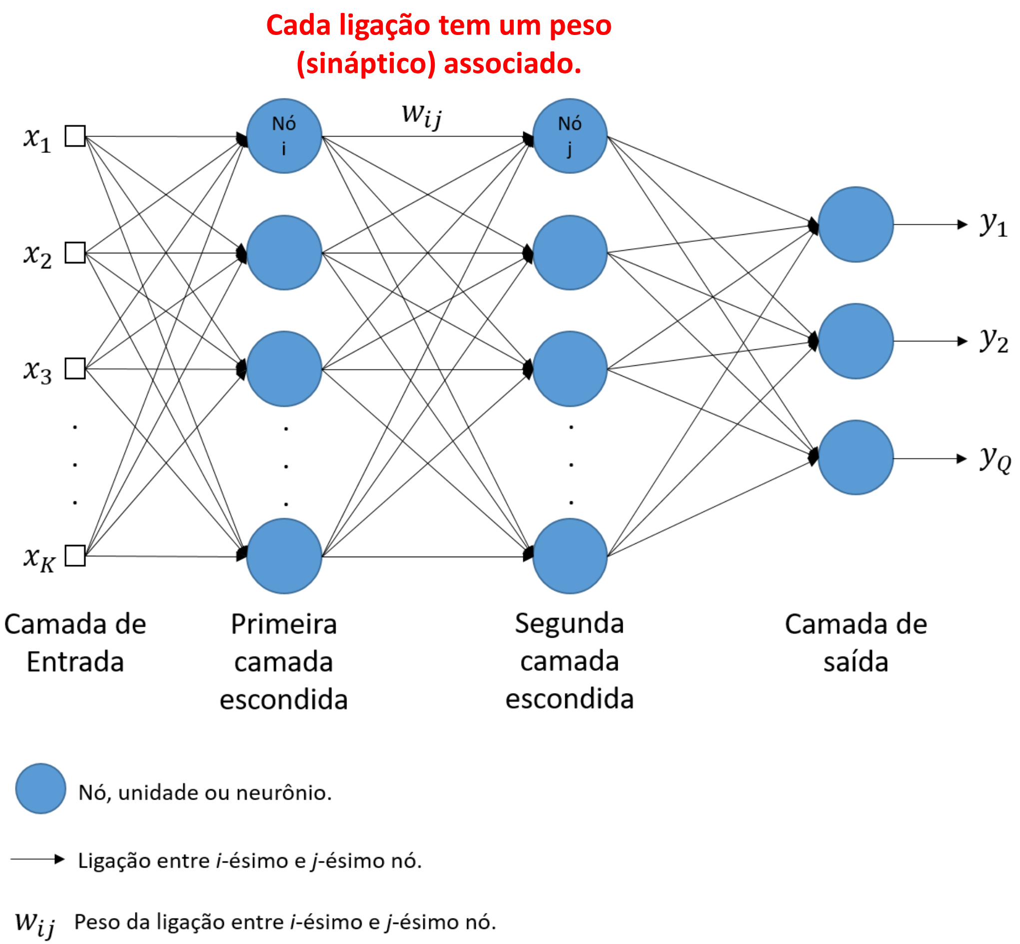
Perceptron de múltiplas camadas





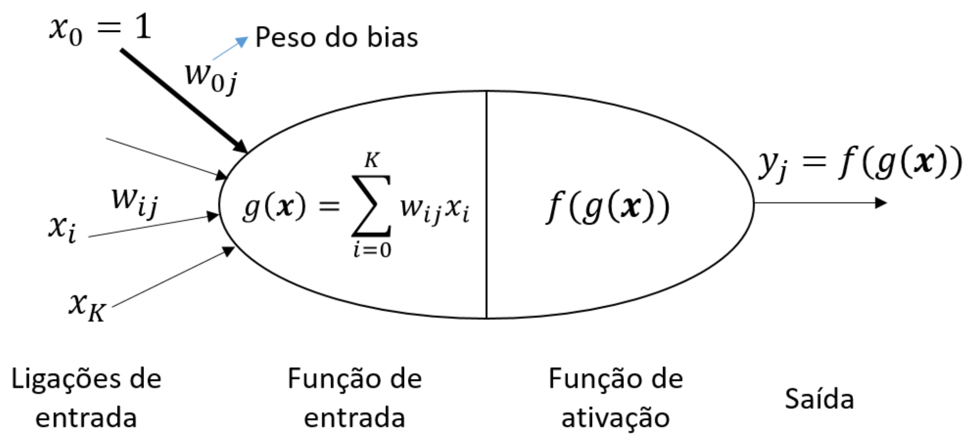
- Em particular, uma MLP com ***uma camada oculta com dois nós*** e ***uma camada de saída com um nó*** pode resolver o problema da lógica XOR.
- Lembrem-se que um único ***perceptron*** não é capaz de realizar essa tarefa.
- Os dois nós da camada oculta ***aprendem separadores lineares*** que são ***combinados***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

Perceptron de múltiplas camadas



- Considerando ***qualquer dois nós da rede***, a ***ligação*** do i-ésimo ***nó*** para o j-ésimo ***nó*** é feita através do ***peso*** w_{ij}.
- A ligação ***propaga*** o ***sinal de saída*** do i-ésimo ***nó*** para o j-ésimo ***nó***.
  - O ***sinal de saída*** do i-ésimo nó 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



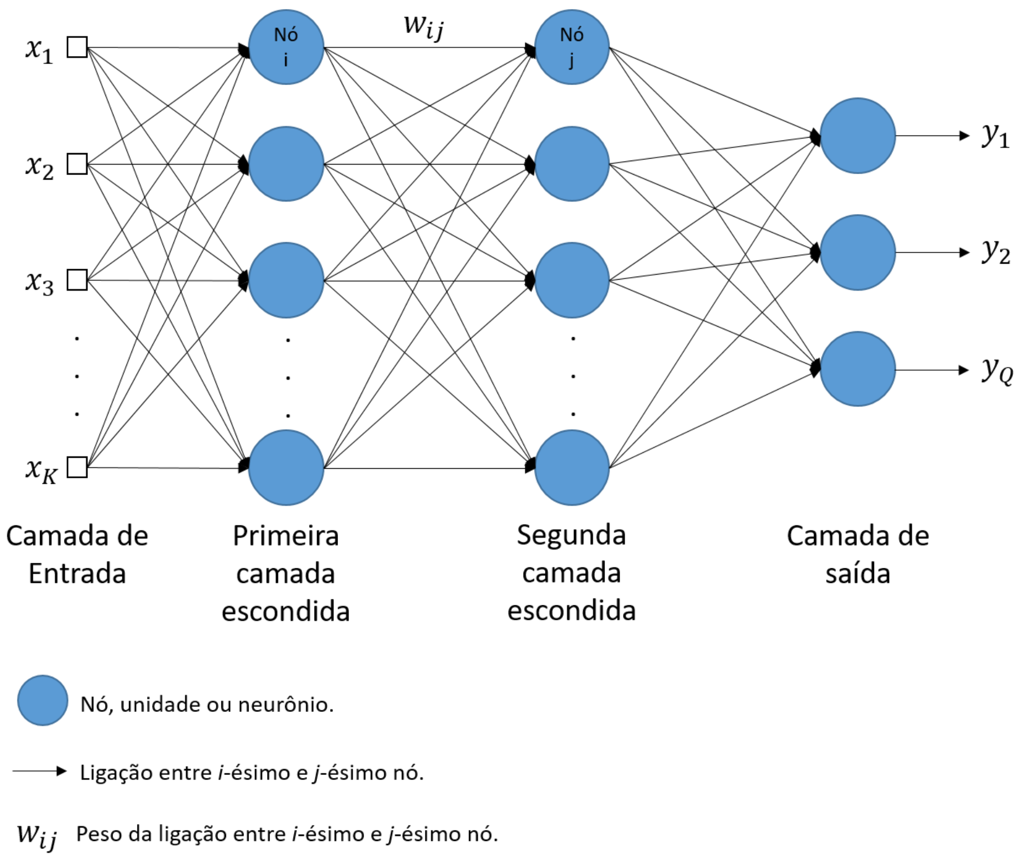
- ***Cada nó*** tem a entrada x_0 (i.e., o atributo de bias) sempre com valor igual a 1 e um peso associado w_{0j}, chamado de ***peso de bias***.
  - Ou seja, a entrada x_0 ***não está conectada a nenhum outro nó***.
- O j-ésimo ***nó*** calcula a ***soma ponderada*** de suas entradas, x_i

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

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

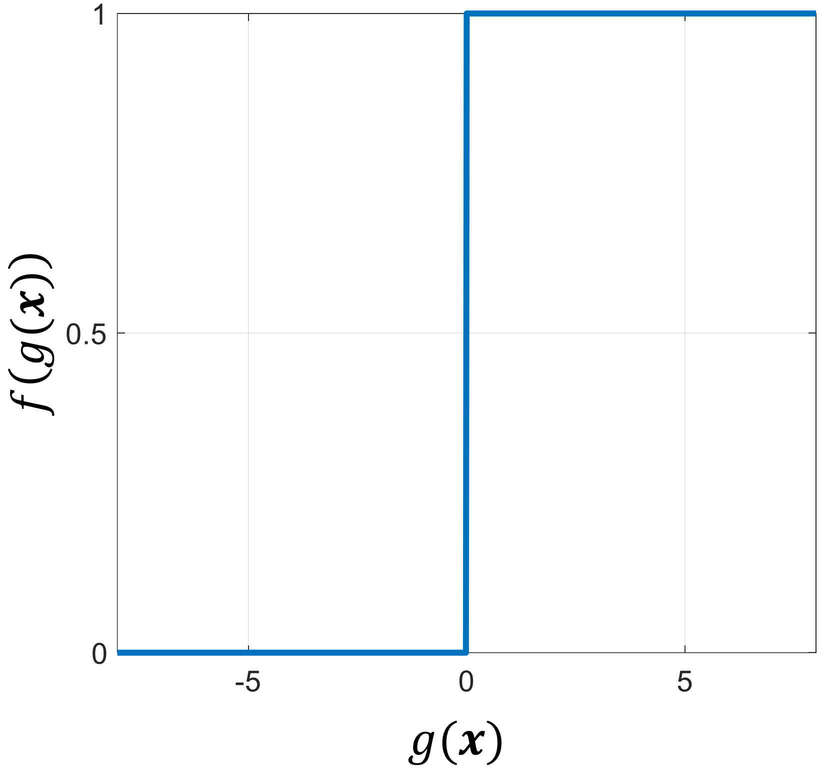
$$y_j = f(g(\boldsymbol{x})).$$

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

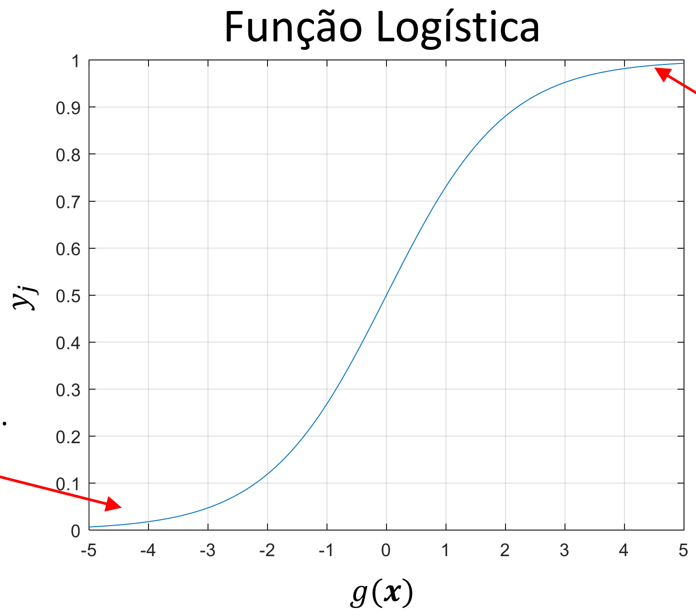
$$\frac{dy_j}{dg(\boldsymbol{x})} = \frac{df(g(\boldsymbol{x}))}{dg(\boldsymbol{x})} = y_j(1 - y_j) \geq 0.$$

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

Função logística e sua derivada

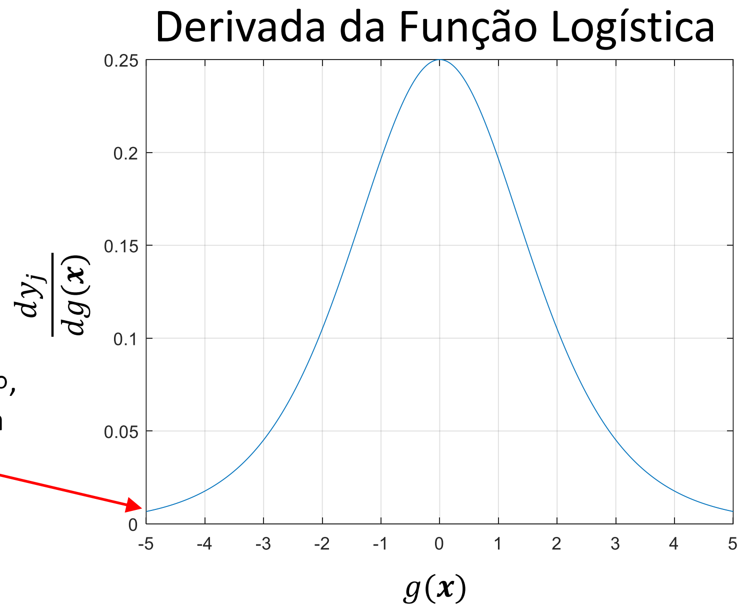
- Percebam que o valor da derivada ***sempre será menor do que 1***, ***sendo no máximo igual a 0.25 quando*** $g(\boldsymbol{x}) = 0$.

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



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

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



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

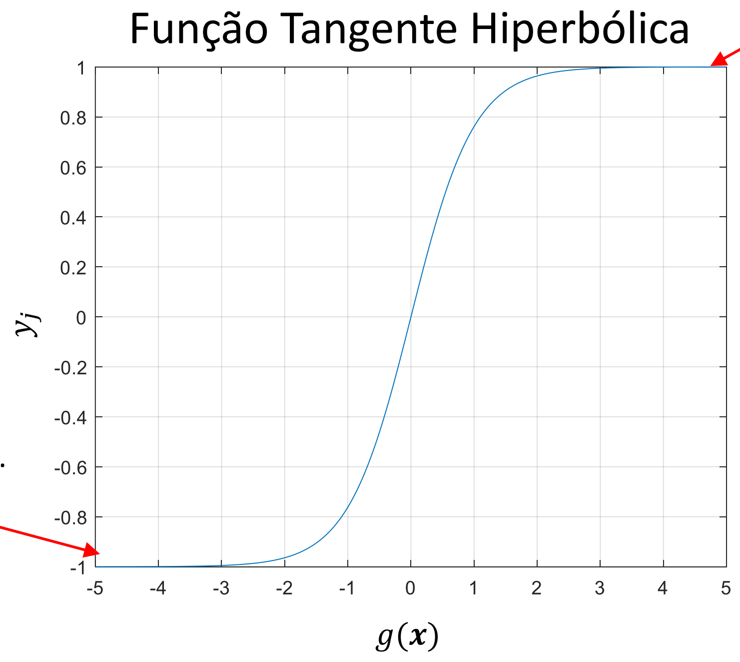
- Sua derivada é dada por

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

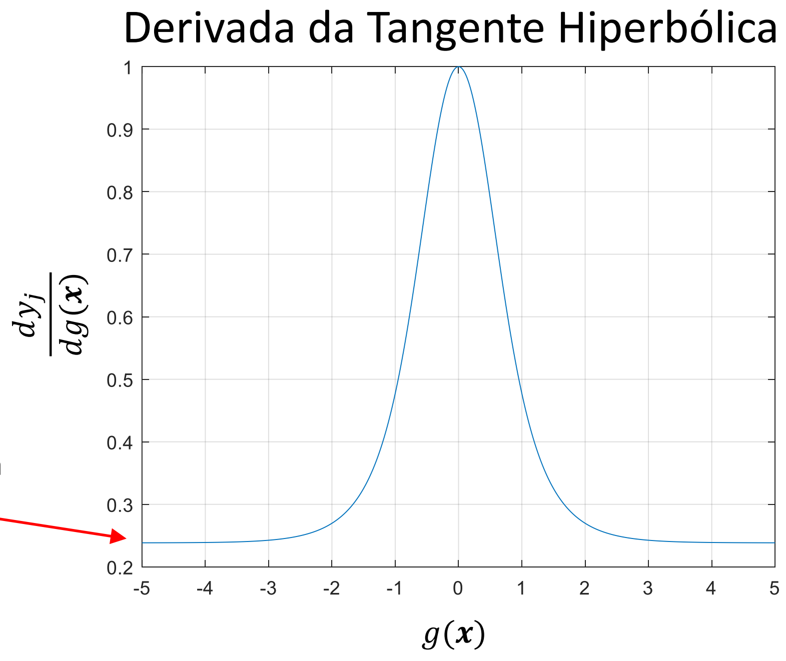
Função tangente hiperbólica e sua derivada

- 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 $y_j \rightarrow 1$
quando $g(\boldsymbol{x}) \rightarrow \infty$.



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



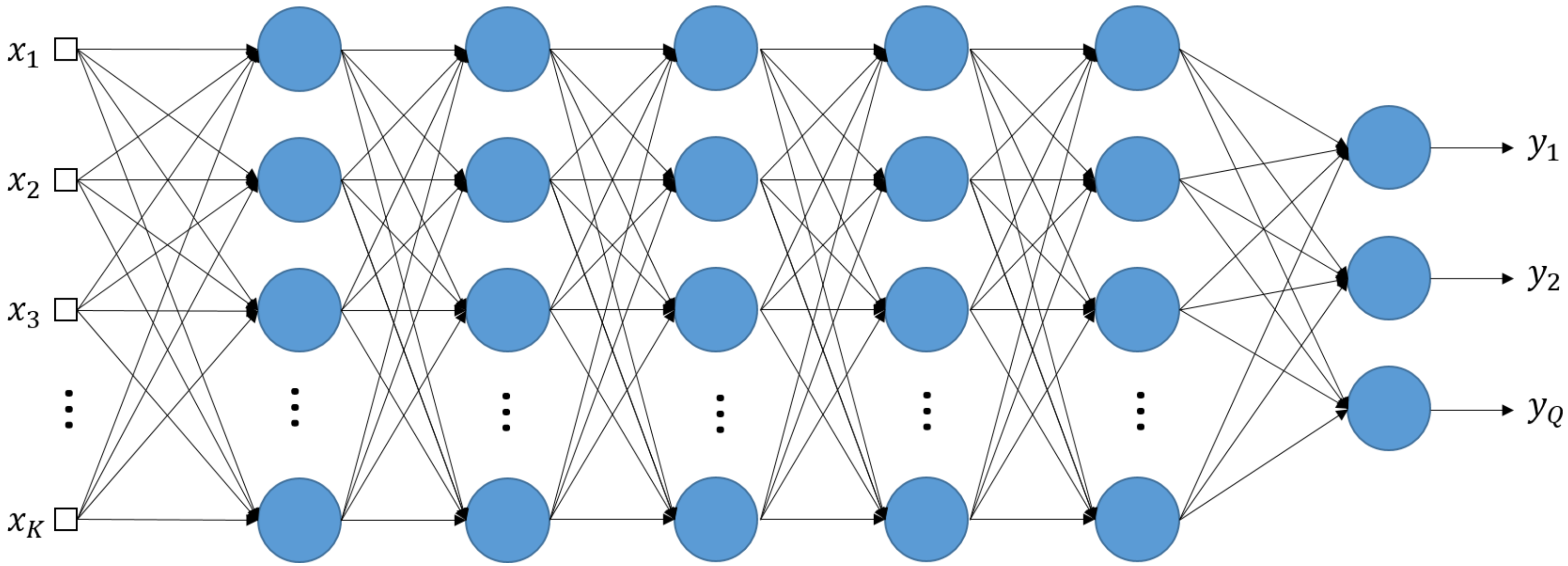
Quando $g(\boldsymbol{x}) \rightarrow -\infty$,
$y_j \rightarrow -1$ e a derivada
tende a 0.

Quando $g(\boldsymbol{x}) \rightarrow \infty$,
$y_j \rightarrow 1$ 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.

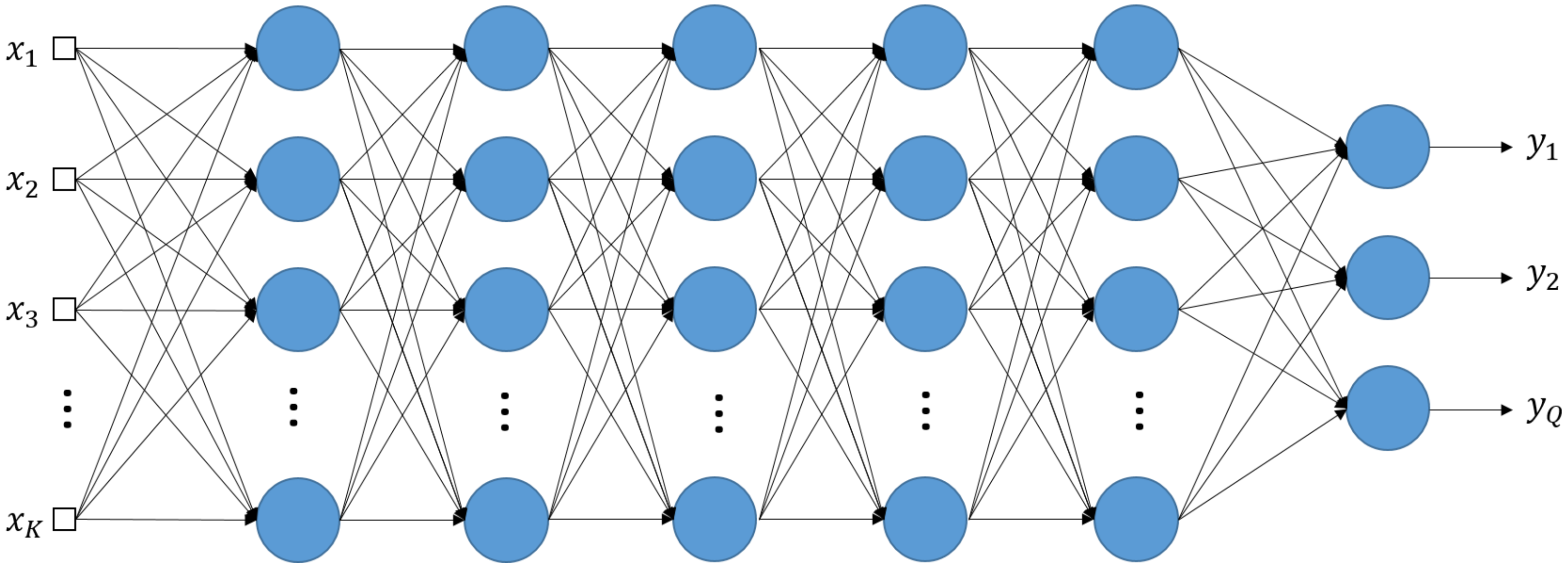
O problema da dissipação do gradiente

- 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***.



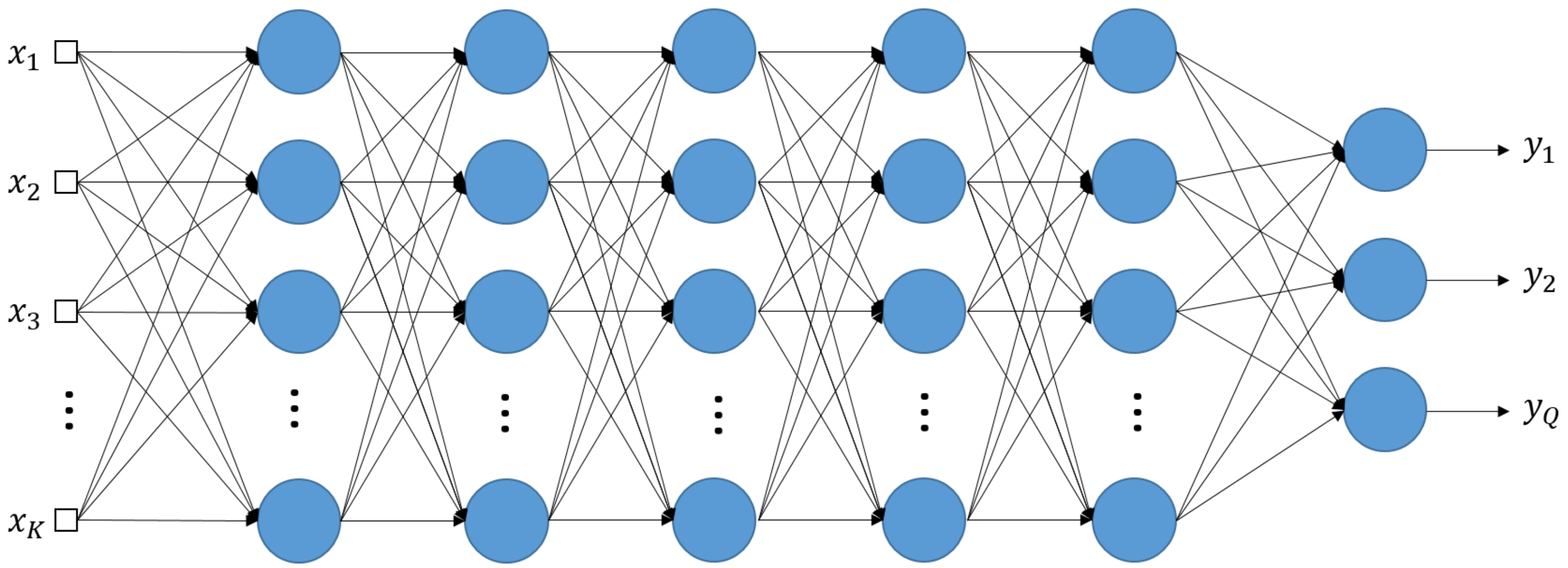
O problema da dissipação do gradiente

- 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***.



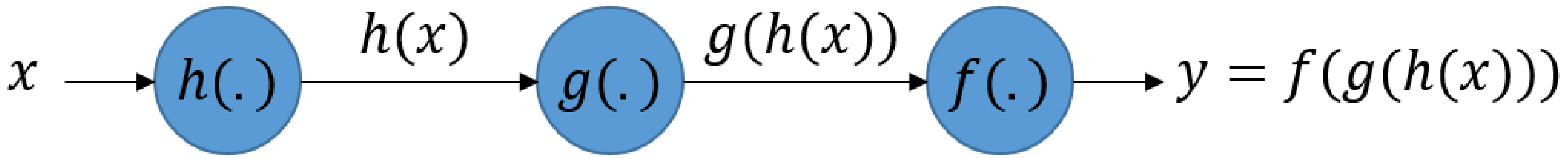
O problema da dissipação do gradiente

- Em suma, problema da dissipação do gradiente faz com que o ***vetor gradiente se torne 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

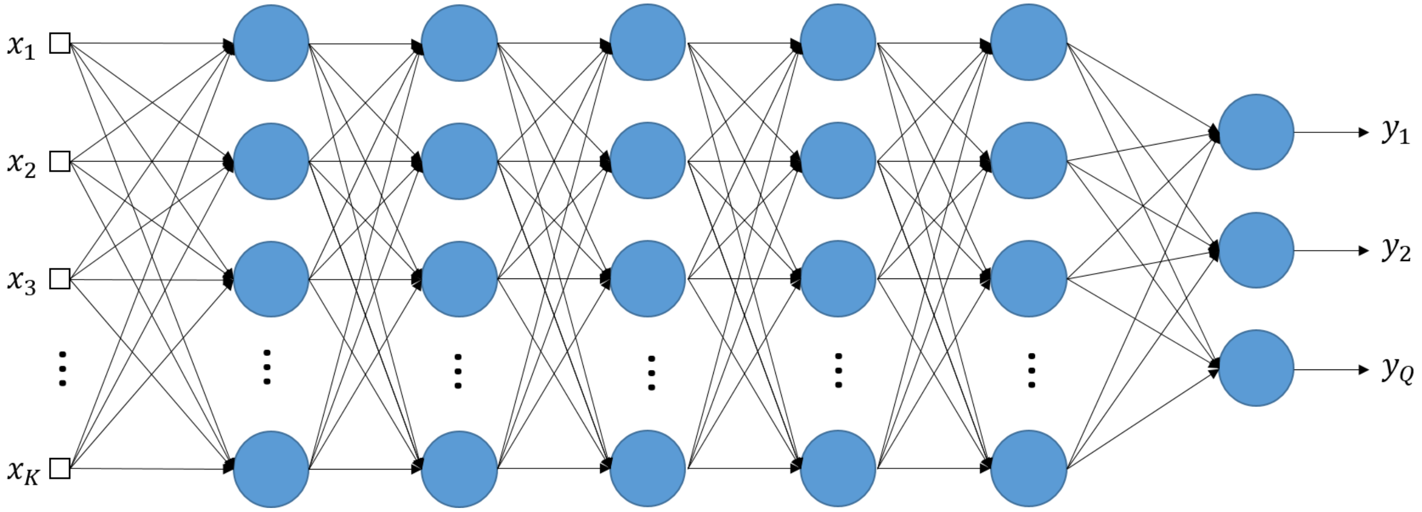
- Em outras palavras, devido à regra da cadeia, o ***vetor gradiente*** para a ***atualização dos pesos de uma dada camada*** da rede inclui o ***produto das derivadas das funções de ativação dos nós desde a camada de saída até a camada desejada***.

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

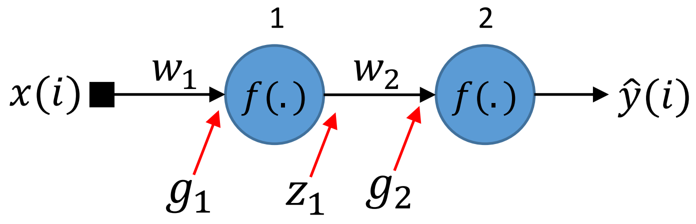
- Lembrem-se que as ***funções de ativação***, como ***tangente hiperbólica*** ou ***logística***, têm derivadas no intervalo de 0 até 1.
- Portanto, a multiplicação de vários termos menores do que 1 tende a 0 conforme o número de camadas da rede aumenta.

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 vetores gradiente das primeiras camadas.
- O que significa que o ***gradiente diminui exponencialmente com*** $\boldsymbol{M}$.
- Assim, os ***nós das camadas iniciais aprendem muito mais lentamente do que os nós das camadas finais***, pois o ***vetor gradiente*** daquelas camadas é ***muito pequeno***, fazendo com que a ***atualização dos pesos também seja pequena.***



Dissipação do gradiente



Considerações:

- 2 x neurônios com função de ativação sigmoide, $f(.)$.
- $g_1 = xw_1 \rightarrow$ entrada (i.e., ativação) do primeiro neurônio.
- $z_1 = f(xw_1) \rightarrow$ saída do primeiro neurônio.
- $g_2 = z_1 w_2 = f(xw_1)w_2 \rightarrow$ entrada (i.e., ativação) do segundo neurônio.
- $\hat{y} = f(f(xw_1)w_2) \rightarrow$ saída do segundo neurônio.
- **Objetivo**: minimizar o erro quadrático médio, $J_e = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(i) - y(i))^2$.

Dissipação do gradiente

- As ***regras de atualização*** dos dois pesos são dadas por

$$w_2 = w_2 - \alpha \frac{\partial J_e}{\partial w_2},$$

$$w_1 = w_1 - \alpha \frac{\partial J_e}{\partial w_1}.$$

- Usando a regra da cadeia, obtemos as derivadas $\frac{\partial J_e}{\partial w_1}$ e $\frac{\partial J_e}{\partial w_2}$

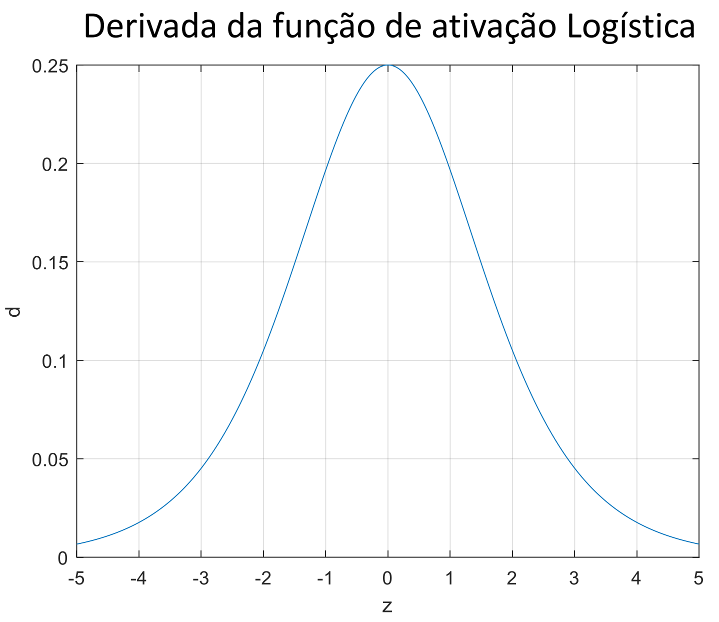
$$\frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2},$$

$$\frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial z_1} \frac{\partial z_1}{\partial g_1} \frac{\partial g_1}{\partial w_1}.$$

Dissipação do gradiente

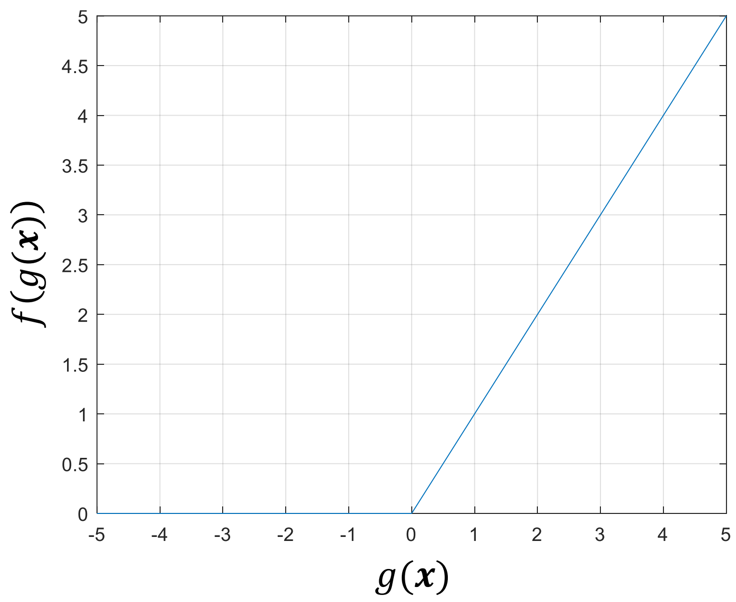
$$\frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2},$$

$$\frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial z_1} \frac{\partial z_1}{\partial g_1} \frac{\partial g_1}{\partial w_1}.$$



- A derivada da função sigmoide é no máximo igual a 0.25.
- Assim, por exemplo, a primeira camada de uma rede neural com M camadas, terá as derivadas parciais da função de erro em relação a seus pesos compostas pela multiplicação de M termos no máximo iguais a 0.25.
- Isso faz com que as primeiras camadas aprendam lentamente ou nem aprendam, pois têm derivadas muito pequenas, tendendo a zero.

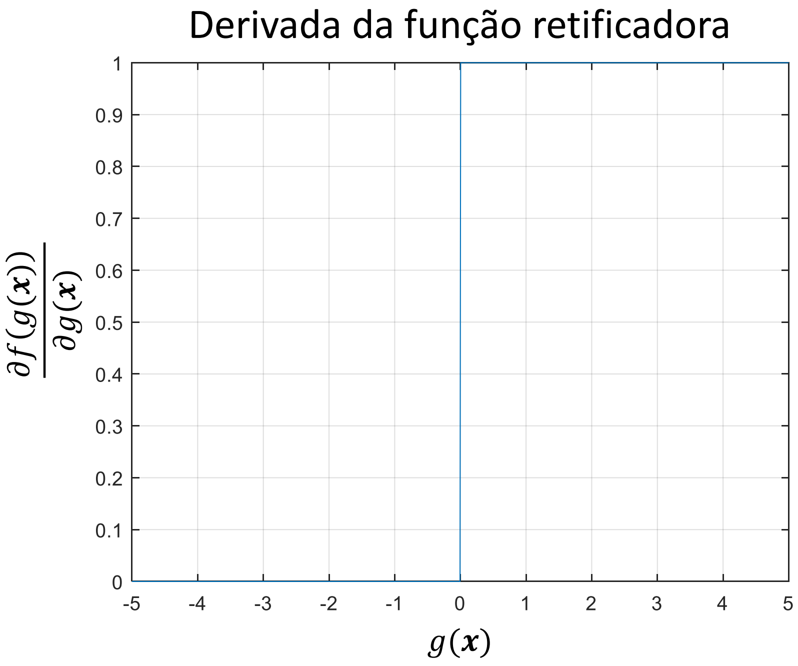
Função de ativação retificadora



$$\hat{y} = f(g(\boldsymbol{x})) = \max(0, g(\boldsymbol{x}))$$

- Com o surgimento das ***redes neurais profundas***, e, consequentemente, do problema do ***desaparecimento do gradiente***, uma outra função de ativação, conhecida como ***Rectified Linear Unit*** (ReLU), passou a ser a bastante utilizada.
- É uma ***função não-linear*** onde sua saída é igual 0 quando $g(\boldsymbol{x}) \leq 0$ e o próprio $g(\boldsymbol{x})$ quando $g(\boldsymbol{x}) > 0$.
- É uma das funções mais amplamente utilizadas em redes neurais.

Função de ativação retificadora

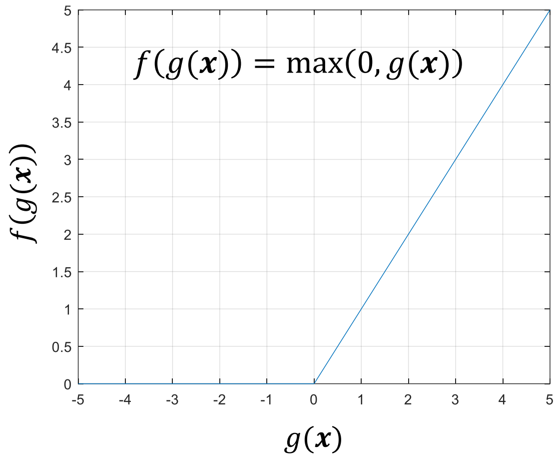


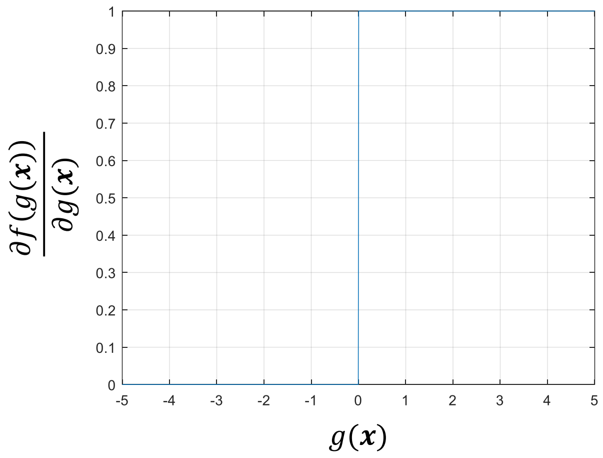
- Suas principais ***vantagens*** são a sua ***simplicidade e eficiência computacional***.
  - Ela e sua derivada ***são mais rápidas de se calcular*** do que as funções logística e tangente hiperbólica.
- Além disso, ajuda a ***minimizar o problema do desaparecimento de gradiente***, pois sua derivada é igual a 1 para $g(\boldsymbol{x}) > 0$.
- Sua derivada é dada por

$$\frac{dy_j}{dg(\boldsymbol{x})} = \frac{df(g(\boldsymbol{x}))}{dg(\boldsymbol{x})} = \begin{cases} 0, \text{se } g(\boldsymbol{x}) < 0 \\ 1, \text{se } g(\boldsymbol{x}) > 0 \end{cases}.$$

- A derivada é indeterminada para $g(\boldsymbol{x}) = 0$.

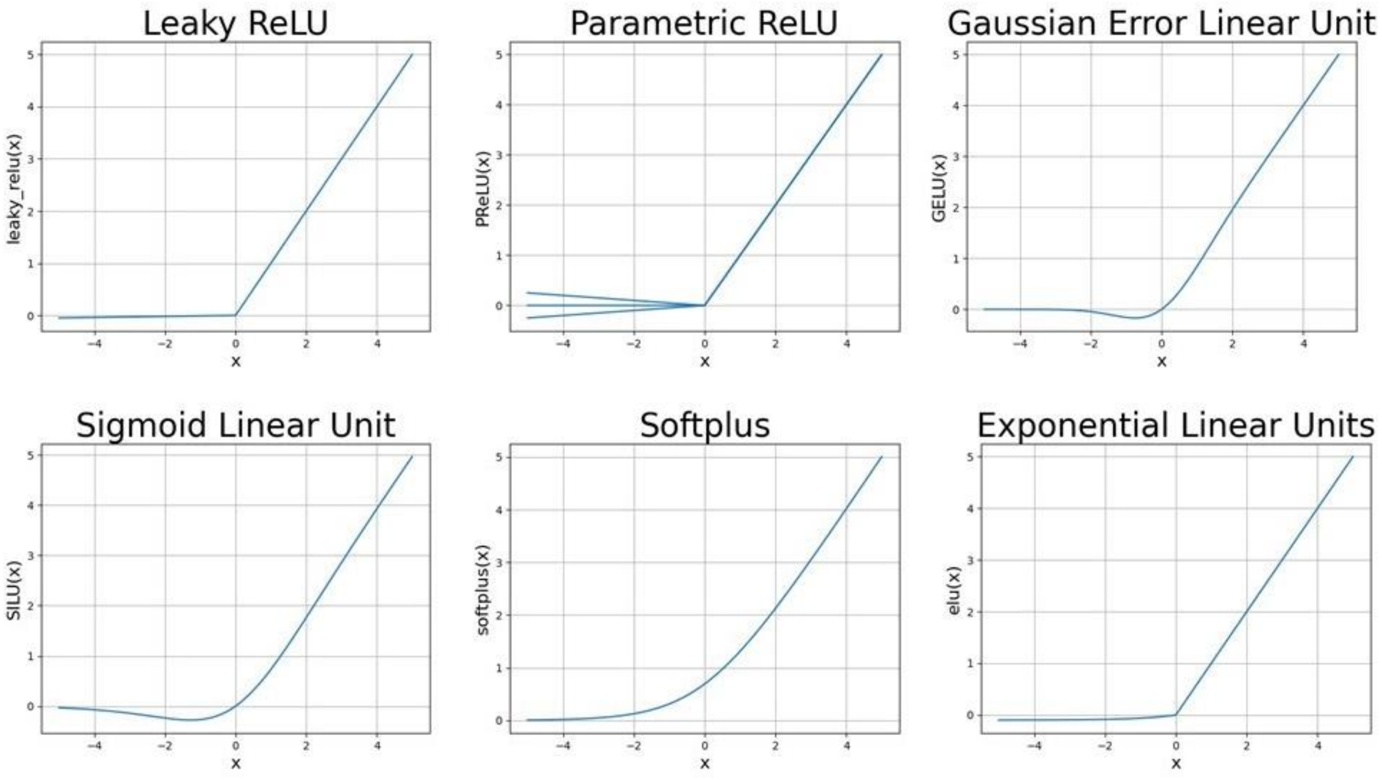
Função de ativação retificadora





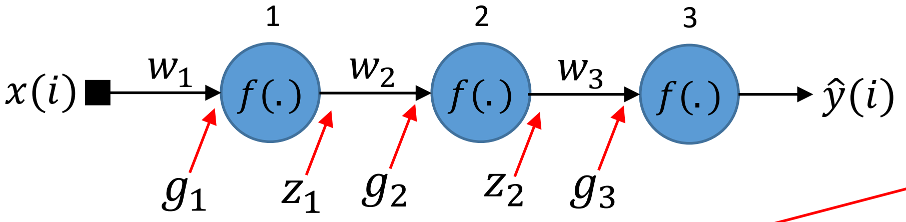
- Uma desvantagem é que ela causa o problema conhecido como ***ReLU agonizante***.
- Esse ***problema ocorre durante o treinamento*** da rede, quando a ativação do nó, $g(\boldsymbol{x})$, é ***negativa***.
- Isso faz com que sua ***saída e***, consequentemente, a ***derivada parcial da função de ativação sejam iguais a 0***.
- Quando isso ocorre, o ***nó não tem seus pesos atualizados*** durante o treinamento, ***permanecendo inalterados***.

Variantes da função de ativação retificadora



- Para resolver o problema das ***ReLUs agonizantes***, usa-se variantes da função ReLU que possuam derivada diferente de zero para $g(\boldsymbol{x}) < 0$, como, por exemplo,
  - Leaky ReLU,
  - Parametric ReLU (PReLU),
  - Gaussian Error Linear Unit (GELU),
  - etc.

Explosão do gradiente



$$\frac{\partial g_3}{\partial z_2} = \frac{\partial f(g_2) w_3}{\partial f(g_2)} = w_3$$

$$\frac{\partial g_2}{\partial z_1} = \frac{\partial f(x w_1) w_2}{\partial f(x w_1)} = w_2$$

$$\frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_3} \frac{\partial g_3}{\partial z_2} \frac{\partial z_2}{\partial g_2} \frac{\partial g_2}{\partial z_1} \frac{\partial z_1}{\partial g_1} \frac{\partial g_1}{\partial w_1}.$$

- Usando ReLUs, reduzimos o problema do desaparecimento do gradiente.
- Porém, caso os pesos sejam inicializados (em geral, de forma aleatória) com valores maiores do que 1, haverá a multiplicação de vários valores assim, podendo resultar em valores de gradiente muito grandes.
- Consequentemente, os pesos da rede podem sofrer atualizações extremamente grandes, o que leva a instabilidades numéricas e a um treinamento ineficaz ou até mesmo à divergência.

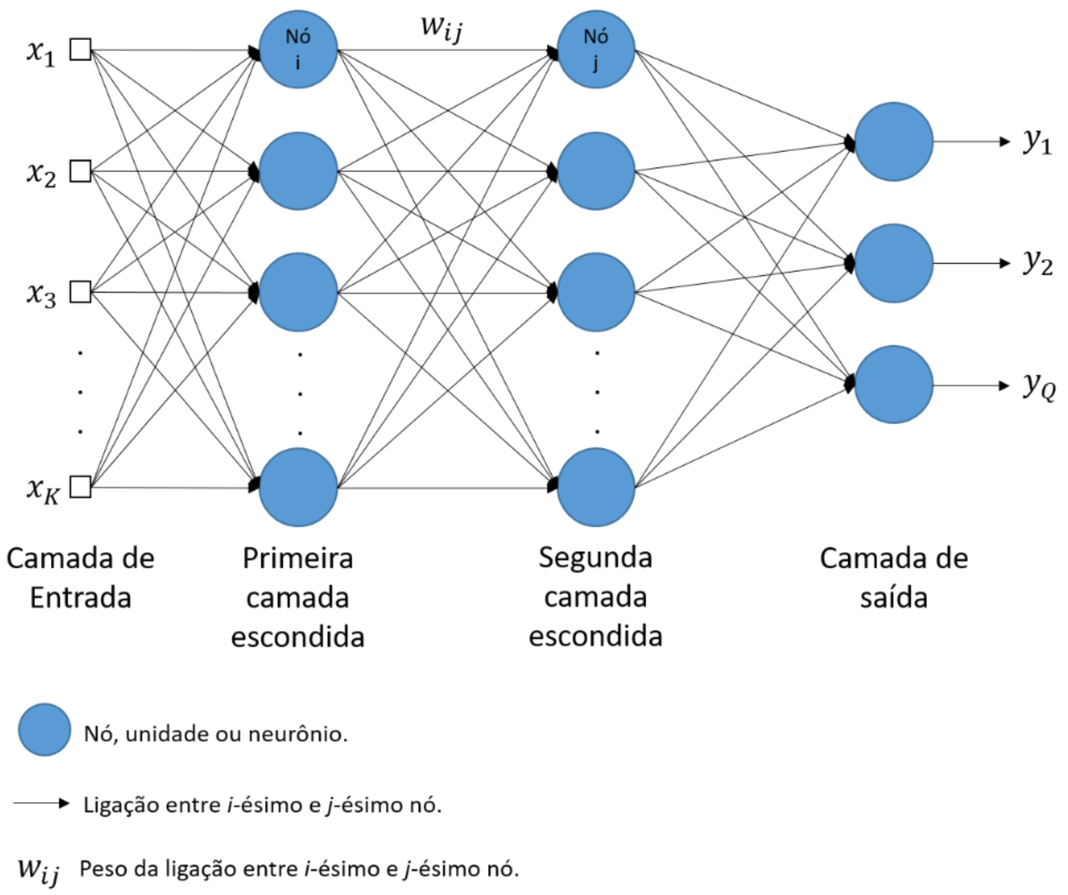
Outras formas de se minimizar a dissipação e a explosão do gradiente

- Além do uso de funções de ativação ReLU ou de suas variantes, outras formas de se minimizar esses problemas são:
  - **Inicialização apropriada dos pesos**: garante que a ***variância das ativações permaneça a mesma ao longo de todas as camadas***. Isso garante que o gradiente retropropagado não tenha multiplicações com valores muito pequenos ou muito grandes em qualquer camada, ajudando a mitigar ambos os problemas.
  - **Normalização de batch**: ***padroniza as ativações*** das camadas da rede e, na sequência, as ***desloca e escalona***, mantendo-as dentro de intervalos que minimizam ambos os problemas.
  - **Poda do gradiente**: *limita (poda) os valores dos gradientes* durante o treinamento para que eles não excedam algum limite pré-definido, mitigando apenas o problema da explosão do gradiente.

Tarefa

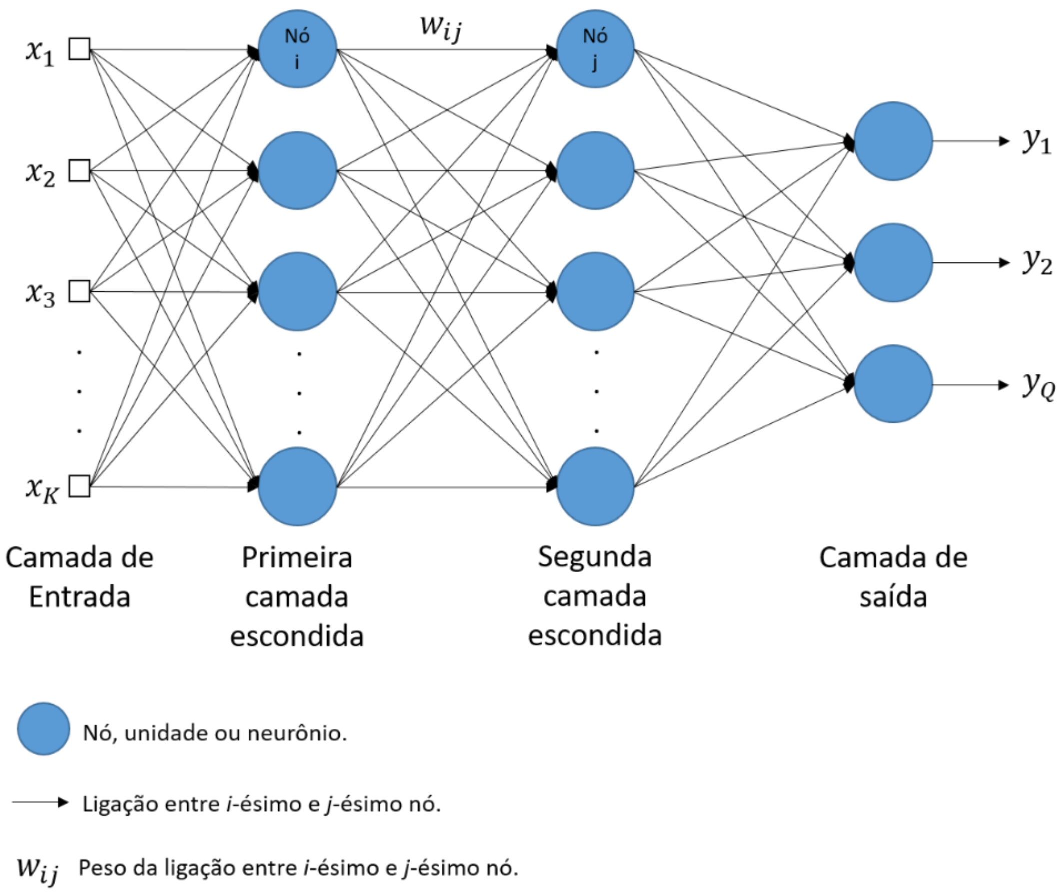
- **Quiz**: “*T320 - Quiz – Redes Neurais Artificiais (Parte III)*” que se encontra no MS Teams.

Conectando neurônios



- Os neurônios de uma rede neural podem ser conectados de forma ***acíclica*** ou ***cíclica***.
- O termo ***acíclico*** se refere a conexões ***sem realimentação***.
- Isso significa que a ***informação flui em uma única direção***, da camada de entrada para a camada de saída.
- A rede ao lado tem conexões ***acíclicas*** e é conhecida como ***rede densa de alimentação direta***.

Conectando neurônios



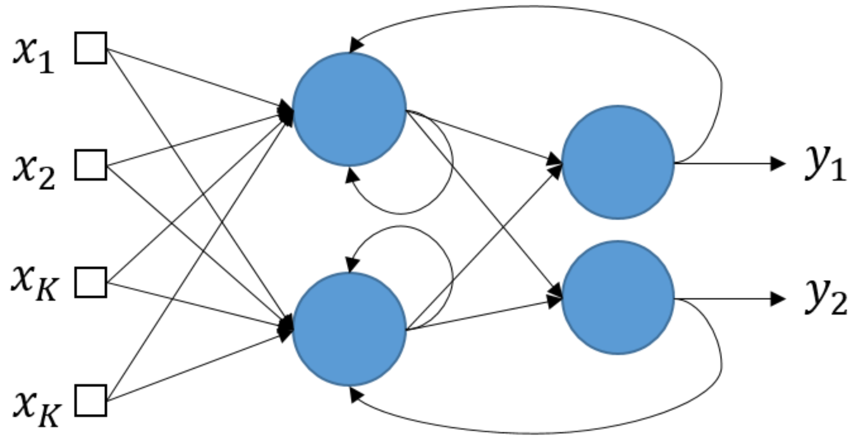
- Esse tipo de rede representa uma ***função de suas entradas e pesos atuais***

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{W}),$$

onde $\boldsymbol{W}$ é a matriz contendo todos os pesos da rede.

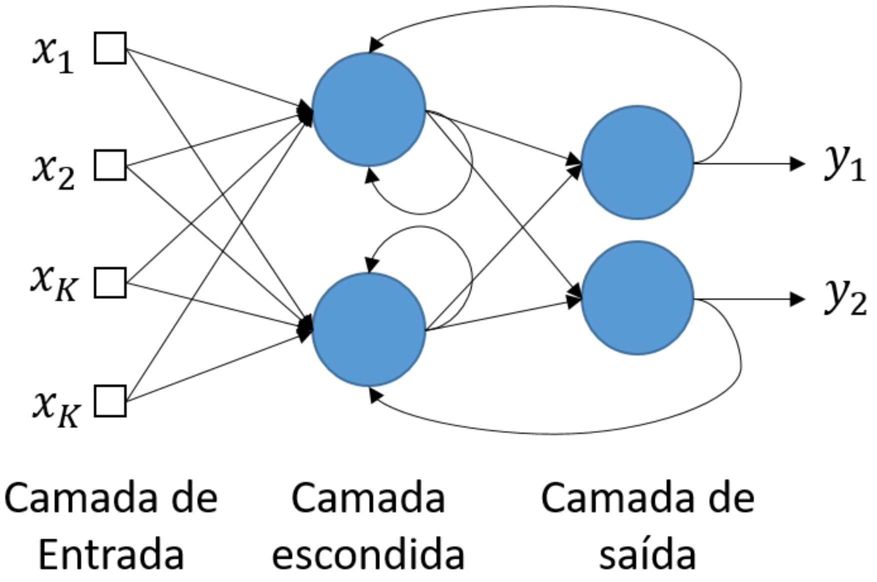
- Portanto, este tipo de rede ***não possui um estado interno, ou seja, memória.***

Conectando neurônios



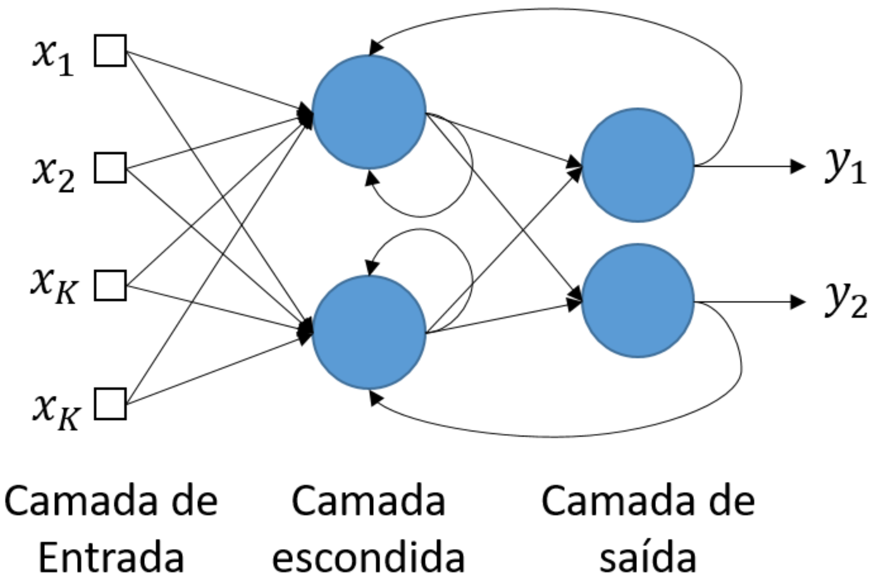
- Já o termo ***cíclico*** se refere a ***conexões que formam ciclos***, permitindo a ***realimentação de informações***.
- Redes com esse tipo de conexão são conhecidas como ***redes recorrentes*** ou ***redes com realimentação***.
- A figura mostra que os nós da rede têm ***conexões em duas direções***, desta forma, o ***sinal percorre a rede nas direções direta e reversa***.

Conectando neurônios



- Esse tipo de rede forma um ***sistema dinâmico*** que pode atingir
  - um estado estável,
  - exibir oscilações
  - ou mesmo um comportamento caótico e divergir.
- Além disso, a saída da rede é ***função de suas entradas e pesos atuais e de seus estados anteriores***, ou seja, de saídas anteriores.

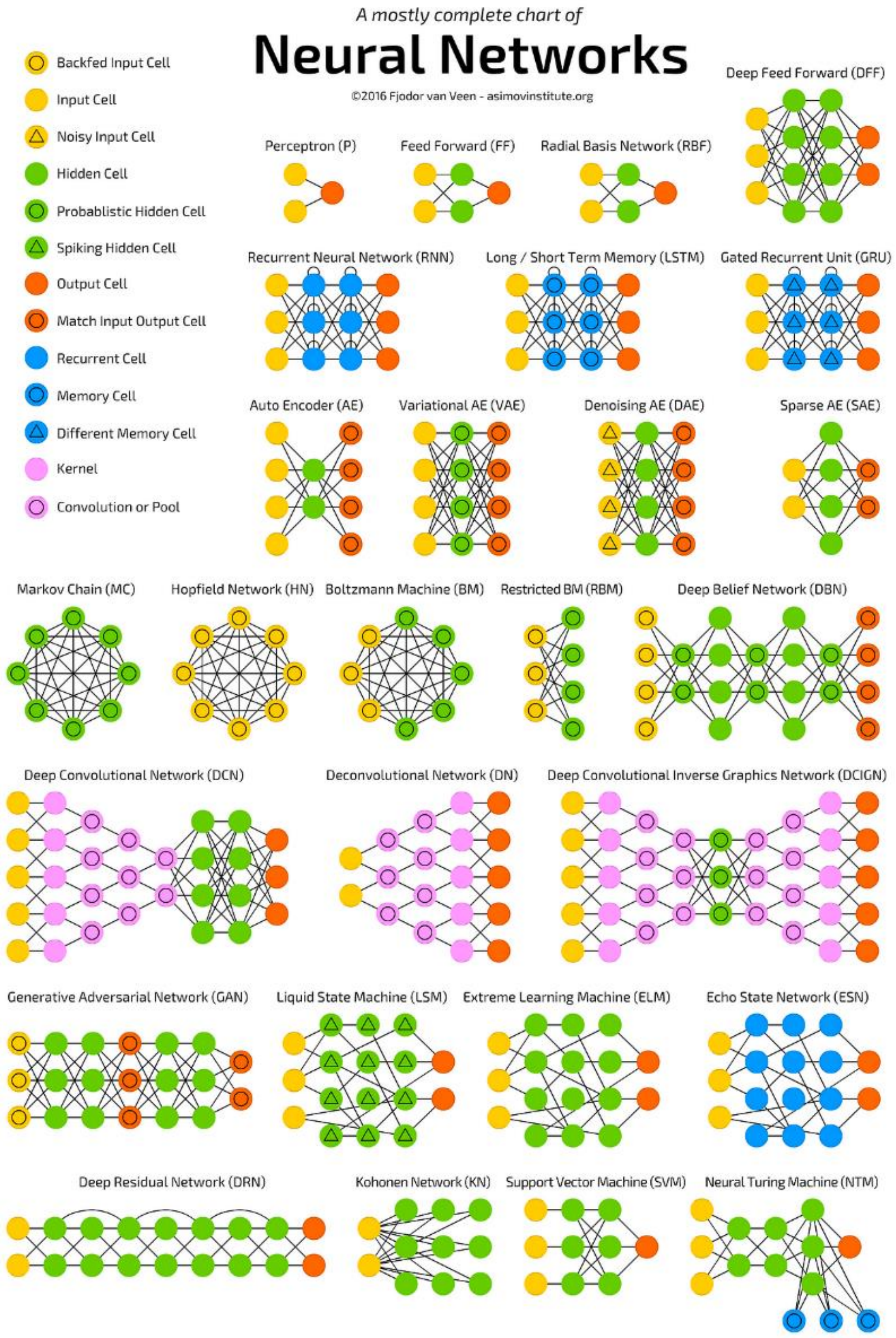
Conectando neurônios



- Portanto, ***redes recorrentes*** possuem ***memória***.
- Essas redes são úteis em tarefas que envolvem ***dependências temporais*** como
  - Previsões de séries temporais (e.g., monitoramento de sinais vitais, preço de ações, etc.) e
  - Processamento de linguagem natural (e.g., conversão de fala em texto, reconhecimento de palavras, respostas a perguntas, etc.).

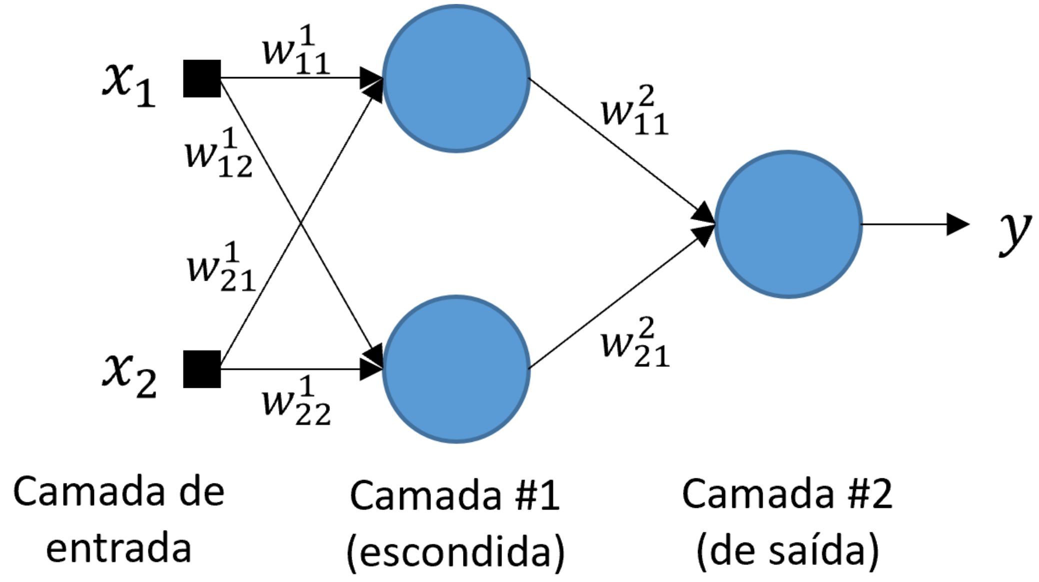
Arquiteturas de redes neurais

- Hoje em dia, existem diversas outras formas de conexão entre camadas e nós que dão origem a uma gama imensa de arquiteturas de redes neurais.
- Um compilado dessas arquiteturas pode ser encontrado em
  - https://www.asimovinstitute.org/author/fjodorvanveen/



Aproximação de funções com redes neurais

Aproximação de funções com redes neurais



- 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 para todos os nós, $\boldsymbol{W} = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}$, $\boldsymbol{w} = \begin{bmatrix} w_{11}^2 \\ w_{21}^2 \end{bmatrix}$ e $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

- Percebam que a saída da rede é dada pelo ***aninhamento*** das saídas de ***funções de ativação não-lineares***.

Aproximação de funções com redes neurais

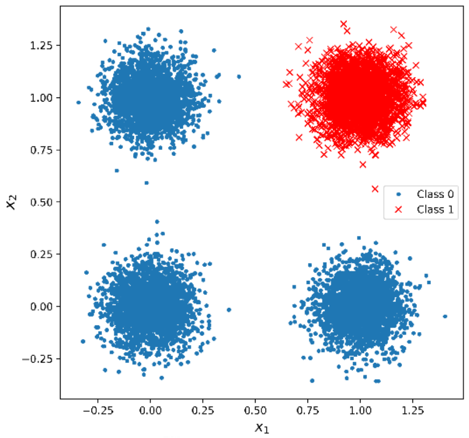
- Portanto, as redes neurais têm a ***capacidade de aproximar funções altamente não-lineares***.
- Essa capacidade ***depende da sua arquitetura***, incluindo o número de camadas, o número de nós (que corresponde à quantidade de pesos) e as funções de ativação empregadas.
  - A ***quantidade de pesos*** de uma rede está associada aos seus ***graus de liberdade***, ou seja, a ***capacidade da rede de aproximar diferentes tipos de funções***.
- Portanto, assim como polinômios, que podem ***aproximar qualquer tipo de função*** (linear ou não linear) devido a seus ***graus de liberdade***, as ***redes neurais podem fazer o mesmo***, bastando apenas que ***encontremos sua complexidade ideal***, ou seja, sua arquitetura.

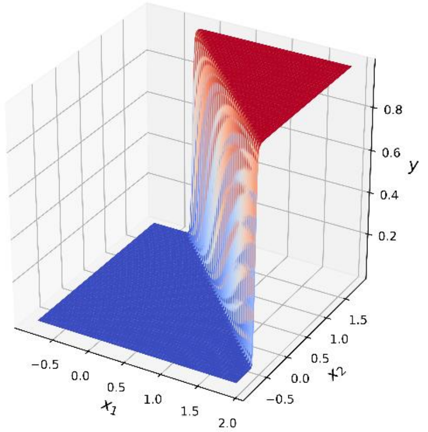
Aproximação de funções com redes neurais

- Por exemplo, uma rede neural com ***uma camada oculta*** com um número ***suficientemente grande de nós*** pode ***aproximar*** praticamente qualquer ***função contínua***.
- Com ***duas camadas ocultas***, até ***funções descontínuas*** podem ser ***aproximadas***.
- Portanto, dizemos que as redes neurais possuem ***capacidade de aproximação universal*** de funções.
- Desta forma, as redes neurais podem resolver ***problemas de regressão e classificação*** e uma grande gama de outros problemas.
- O desafio é encontrar a arquitetura ideal para a aproximação.
- Veremos alguns exemplos desta capacidade de aproximação a seguir.

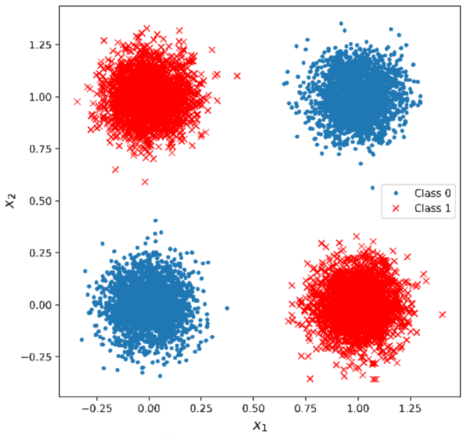
Aproximação universal de funções em problemas de classificação

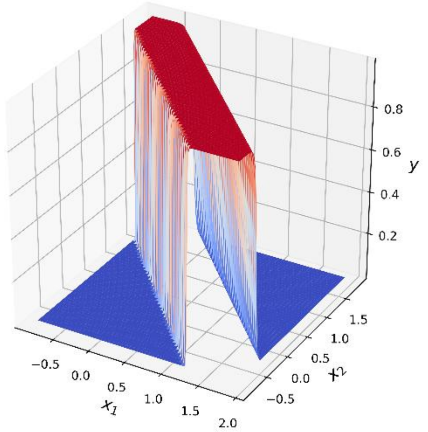
Função AND: MLP sem camada escondida, com apenas um neurônio na camada de saída. Total: 1 nó.



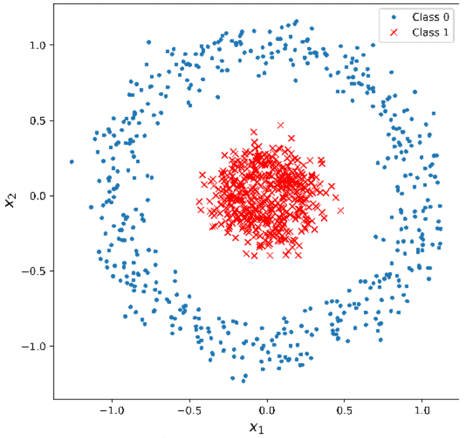


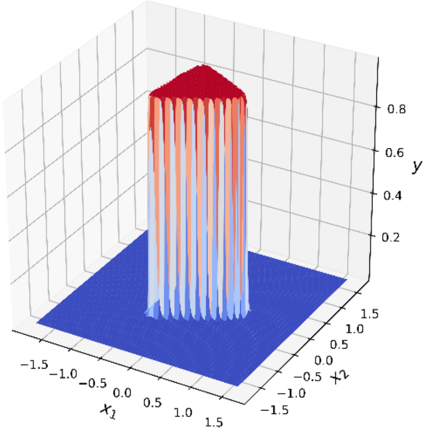
Função XOR: MLP com 1 camada escondida com 2 nós mais 1 nó na camada de saída. Total: 3 nós.





Círculos concêntricos: MLP com 1 camada escondida com 4 nós mais 1 nó na camada de saída. Total: 5 nós.



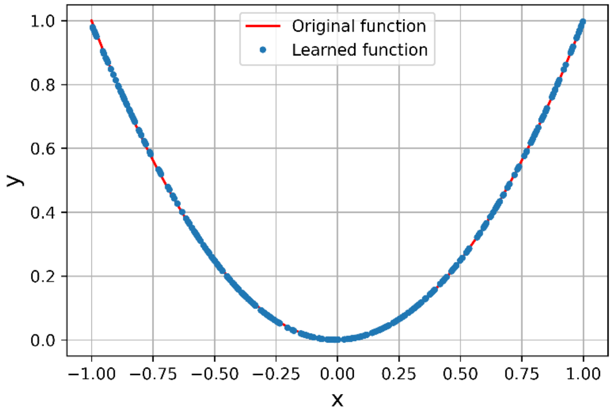


- Fig. 1: Um nó aproxima uma função de limiar suave.
- Fig. 2: Combinando duas funções de limiar suave com direções opostas, podemos obter uma função com formato de onda.
- Fig. 3: Combinando duas ondas perpendiculares, nós obtemos uma função com formato triangular.

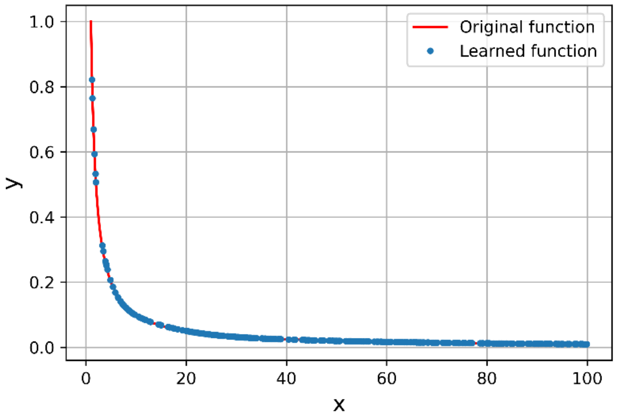
[Exemplo: function_approximation_classification.ipynb]

Aproximação universal de funções em problemas de regressão

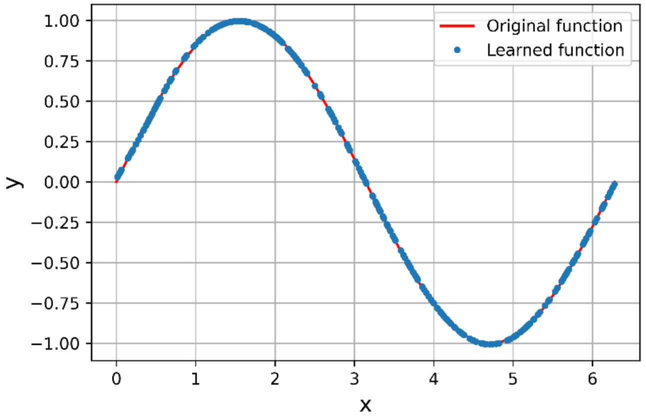
$$f(x) = x^2$$



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



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



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

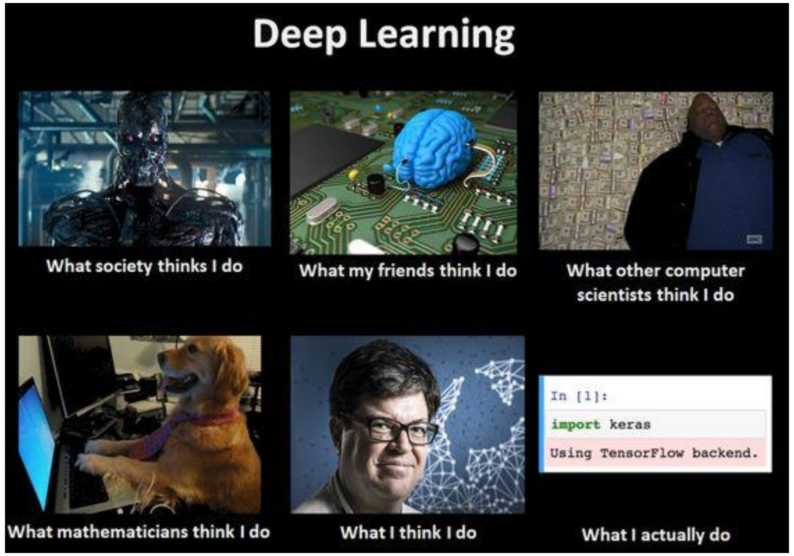
Exemplo: function_approximation_regression.ipynb

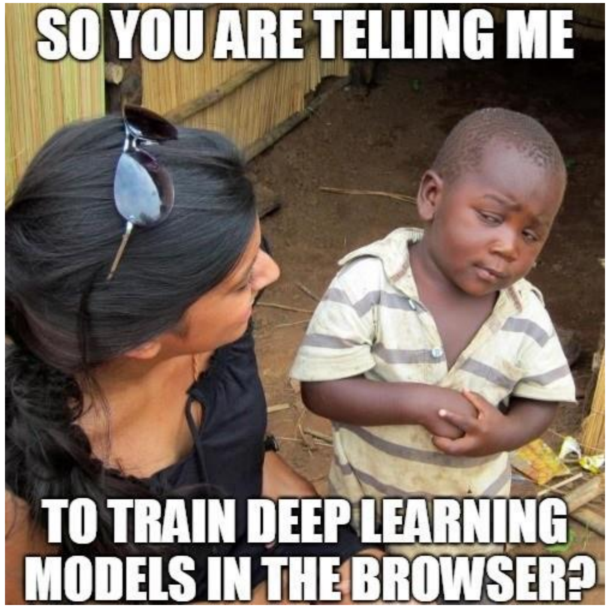
Tarefas

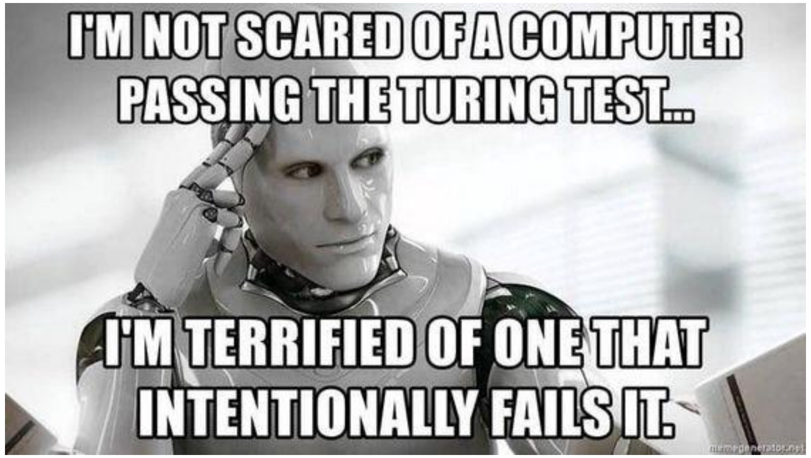
- **Quiz**: *"T320 - Quiz – Redes Neurais Artificiais (Parte IV)"* que se encontra no MS Teams.
- **Exercício Prático**: **Laboratório #7**.
  - Pode ser baixado do MS Teams ou do GitHub.
  - Pode ser respondido através do link acima (na nuvem) ou localmente.
  - Instruções para resolução e entrega dos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

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

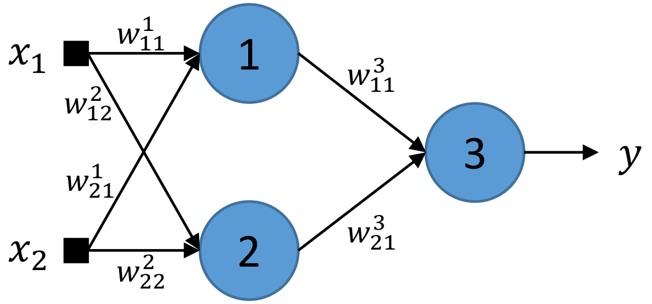

SO YOU ARE TELLING ME
TO TRAIN DEEP LEARNING MODELS IN THE BROWSER?

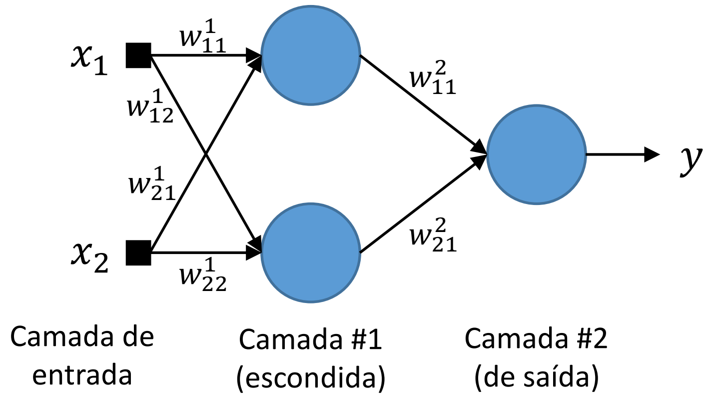

I'M NOT SCARED OF A COMPUTER PASSING THE TURING TEST...
I'M TERRIFIED OF ONE THAT INTENTIONALLY FAILS IT.

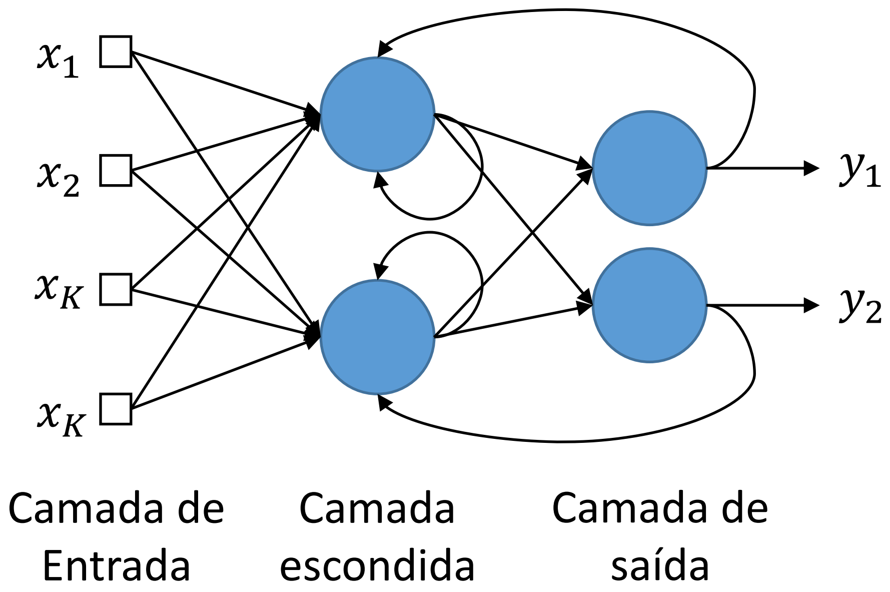

Dog
I NEED GPU FOR MY DUMB NEURAL NETWORK


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
3
w_{11}^3
w_{21}^3
y

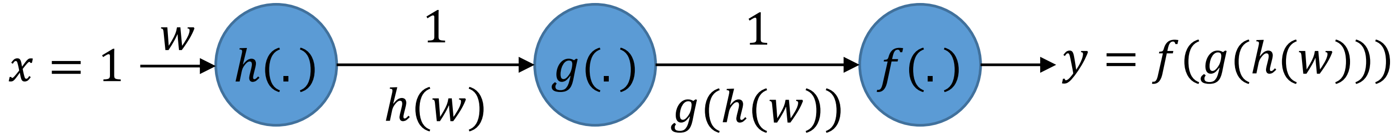

x_1
x_2
w_{11}^1
w_{12}^1
w_{21}^1
w_{22}^1
w_{11}^2
w_{21}^2
y
Camada de entrada
Camada #1 (escondida)
Camada #2 (de saída)

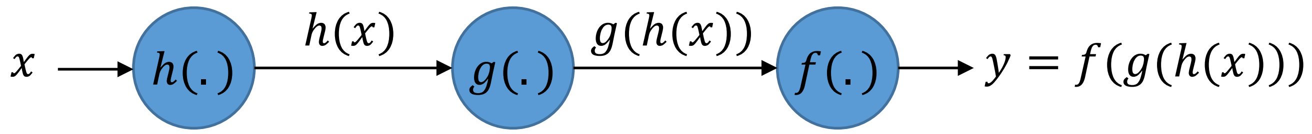

x_1
x_2
x_K
x_K
y_1
y_2
Camada de Entrada
Camada escondida
Camada de saída

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