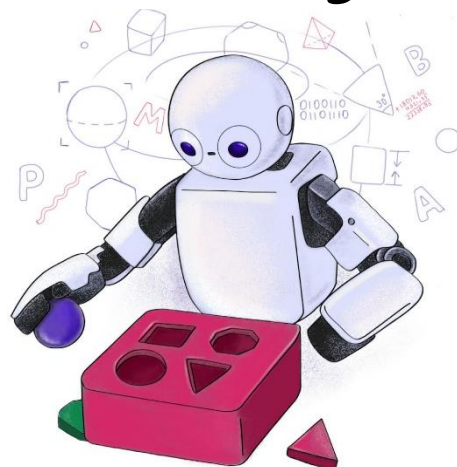


# T320 - Introdução ao Aprendizado de Máquina II: *Redes Neurais Artificiais (Parte II)*



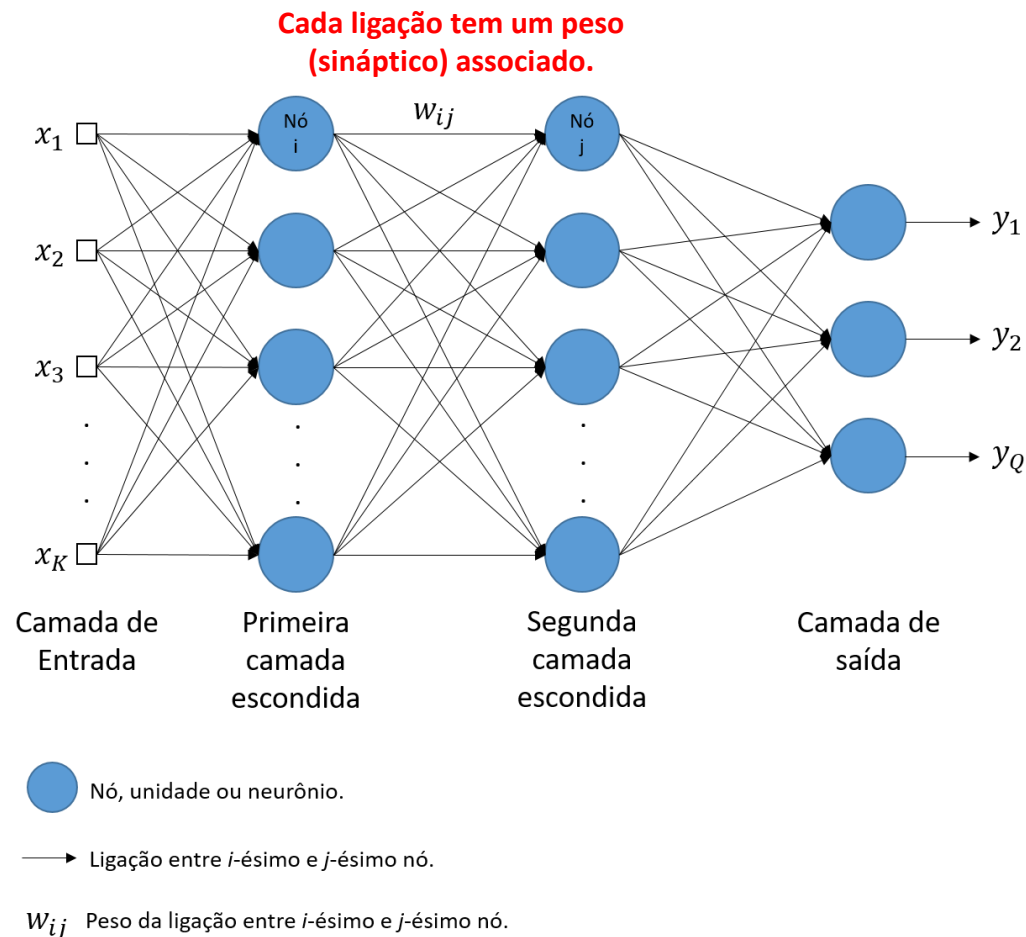
**Inatel**

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

# Recapitulando

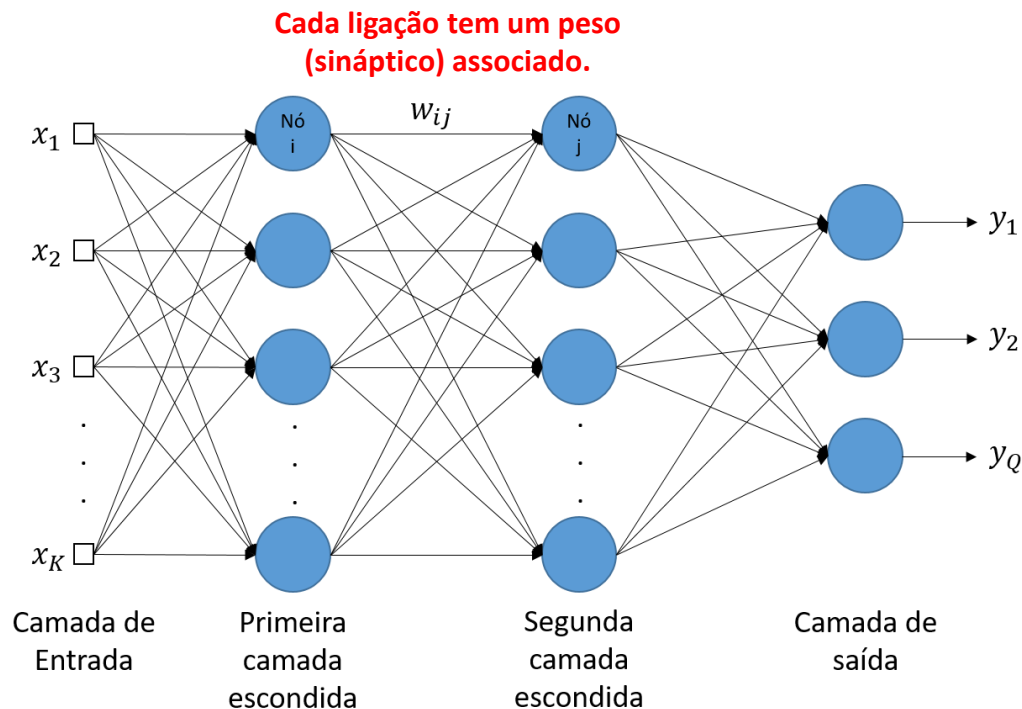
- Fizemos uma analogia entre um neurônio e os modelos de McCulloch e Pitts e do Perceptron.
- Vimos a evolução do modelo de McCulloch e Pitts para o Perceptron.
- Aprendemos suas características, diferenças e como ambos funcionam.
- Verificamos que um Perceptron é semelhante ao regressor logístico.
-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



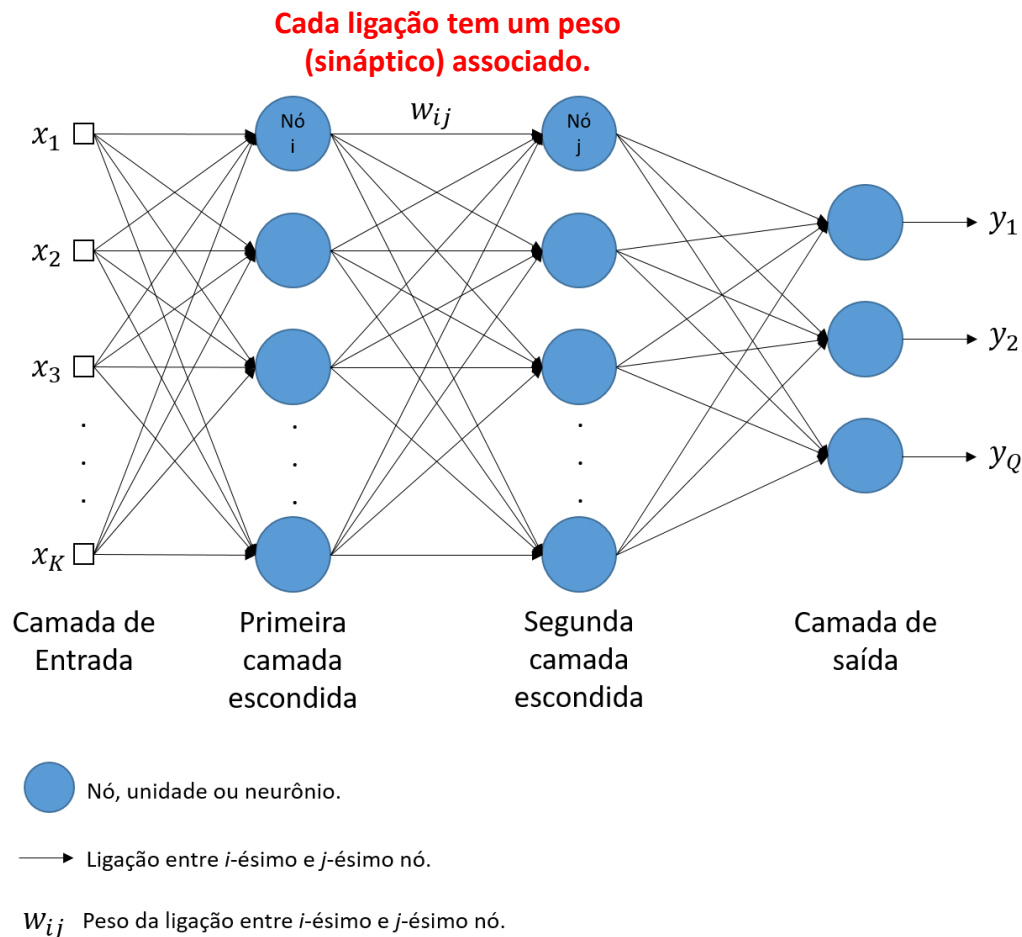
- Algumas das **limitações dos perceptrons** (e.g., classificação apenas de classes linearmente separáveis) podem ser **superadas adicionando-se camadas intermediárias de perceptrons**.
- As camadas intermediárias são também chamadas de **ocultas** ou **escondidas**.

● Nó, unidade ou neurônio.

→ Ligação entre  $i$ -ésimo e  $j$ -ésimo nó.

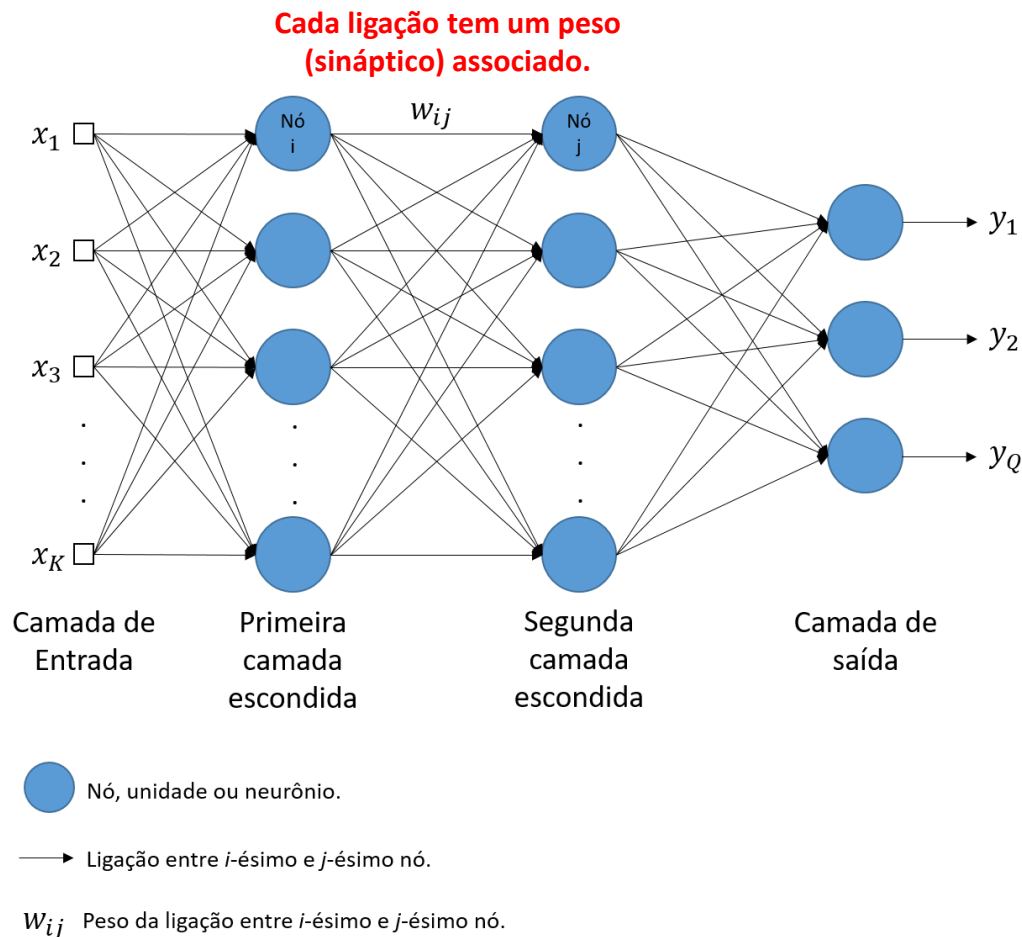
$w_{ij}$  Peso da ligação entre  $i$ -ésimo e  $j$ -ésimo nó.

# Perceptron de múltiplas camadas



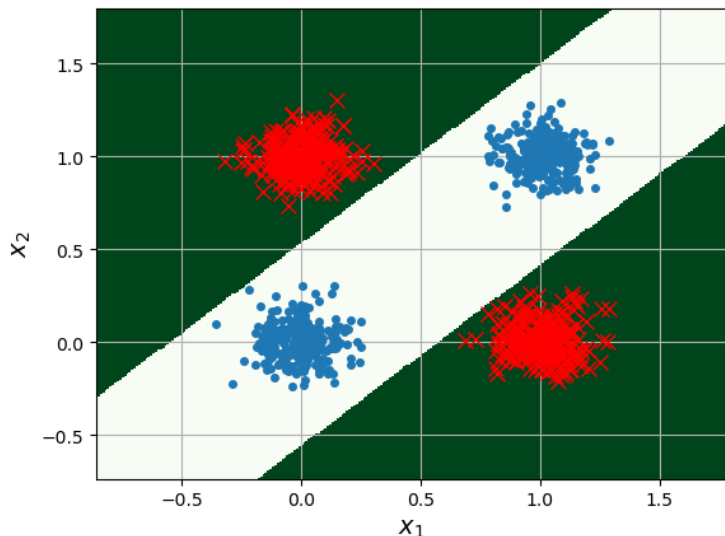
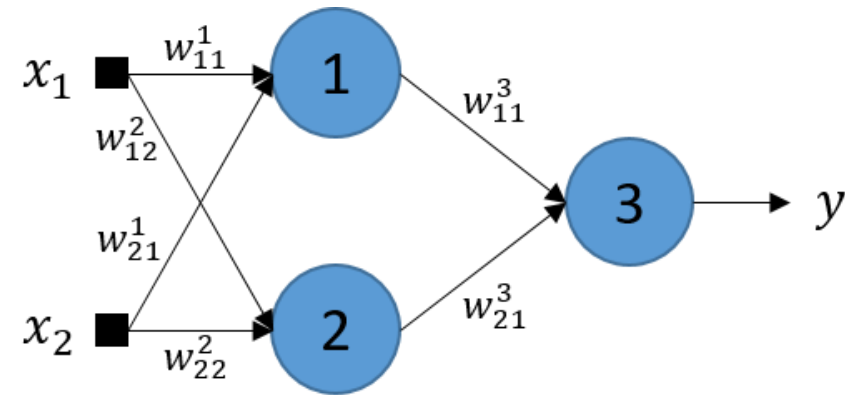
- A rede ao lado é do tipo **densamente conectada** e de **alimentação direta**.
  - Cada uma das saídas de uma camada se conecta a todos os nós da camada seguinte através de pesos sinápticos.
  - Os dados fluem através da rede em uma única direção, da camada de entrada para a camada de saída, sem ciclos 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



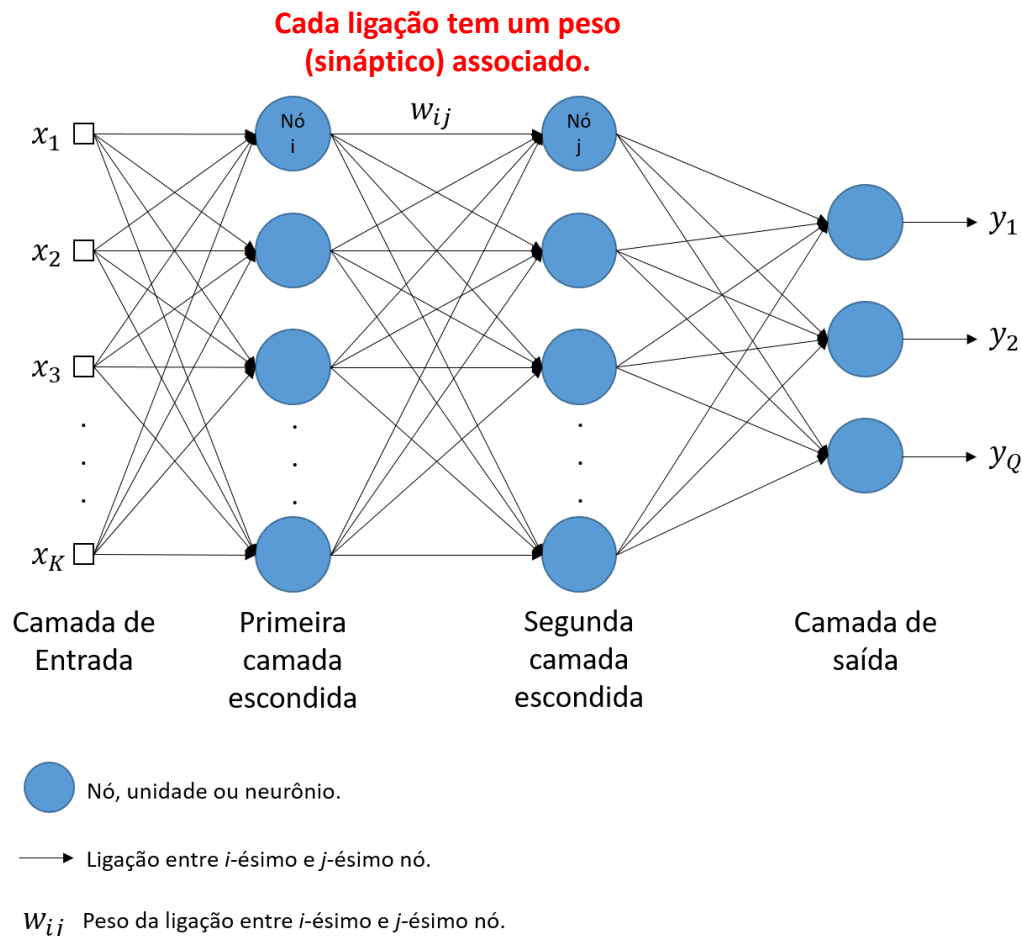
- As RNAs são o coração do ***deep learning ou aprendizado profundo***.
- O termo "***profundo***" vem fato de que essas redes podem possuir muitas camadas ocultas.
- Em geral, quando uma RNA tem duas ou mais camadas ocultas, ela pode ser chamada de ***rede neural profunda*** (ou em inglês, ***Deep Neural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

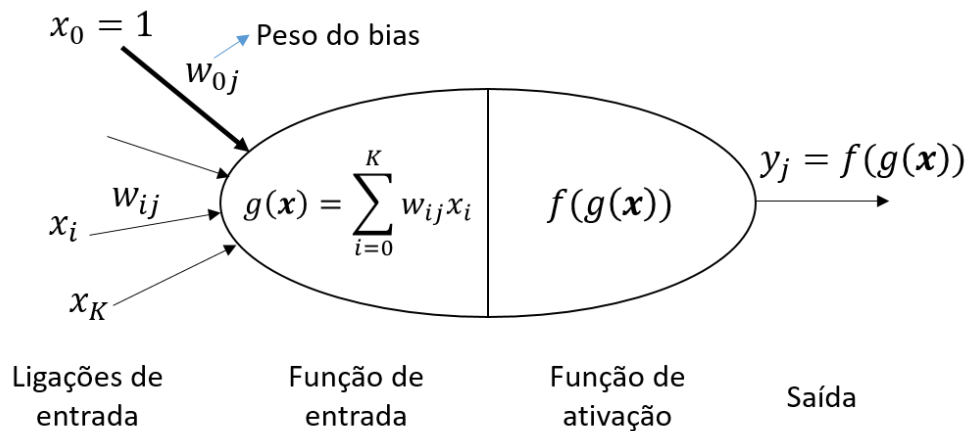
# Perceptron de múltiplas camadas



- 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



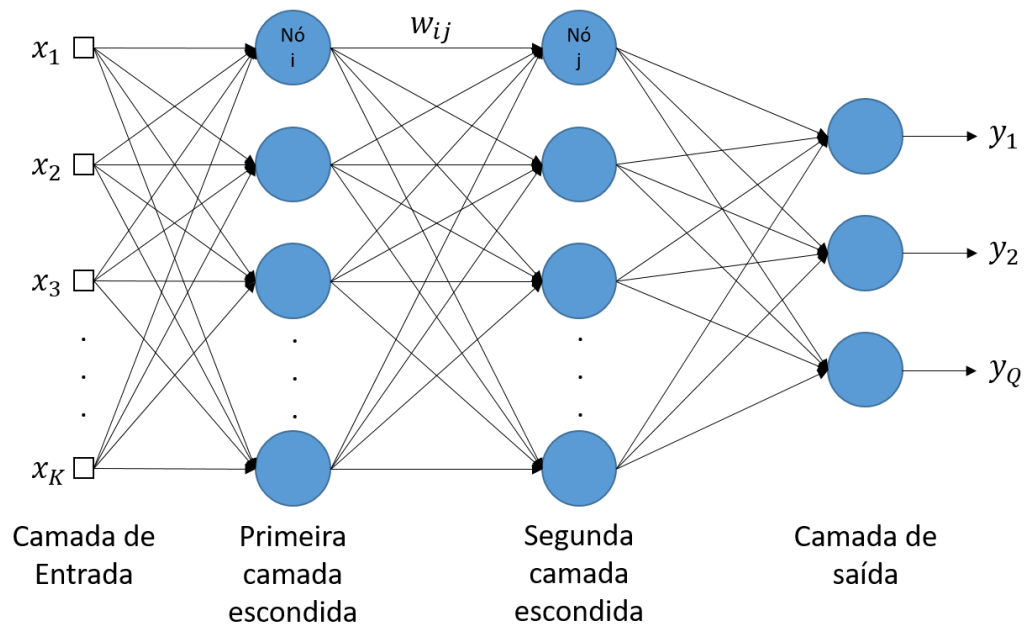
- Cada **nó** tem a entrada  $x_0$  (i.e., o atributo de bias) sempre com valor igual a 1 e um peso associado  $w_{0j}$ , chamado de **peso de bias**.
  - Ou seja, a entrada  $x_0$  **não está conectada a nenhum outro nó**.
- O  $j$ -ésimo **nó** calcula a **soma ponderada** de suas entradas,  $x_i$

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

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

$$y_j = f(g(x)).$$

# Perceptron de múltiplas camadas



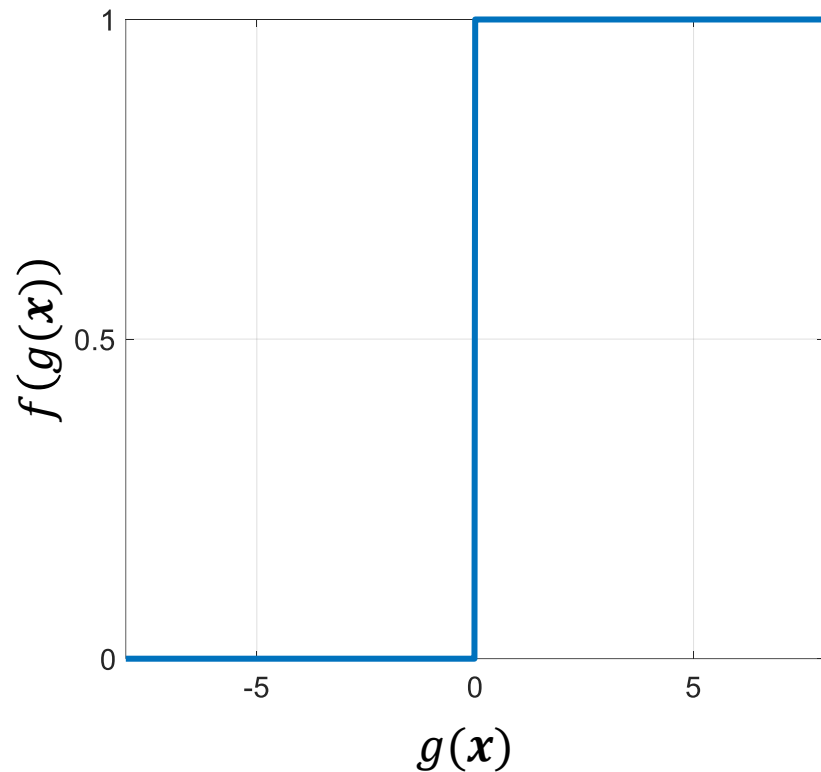
● Nó, unidade ou neurônio.

→ Ligação entre  $i$ -ésimo e  $j$ -ésimo nó.

$w_{ij}$  Peso da ligação entre  $i$ -ésimo e  $j$ -ésimo nó.

- 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

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

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

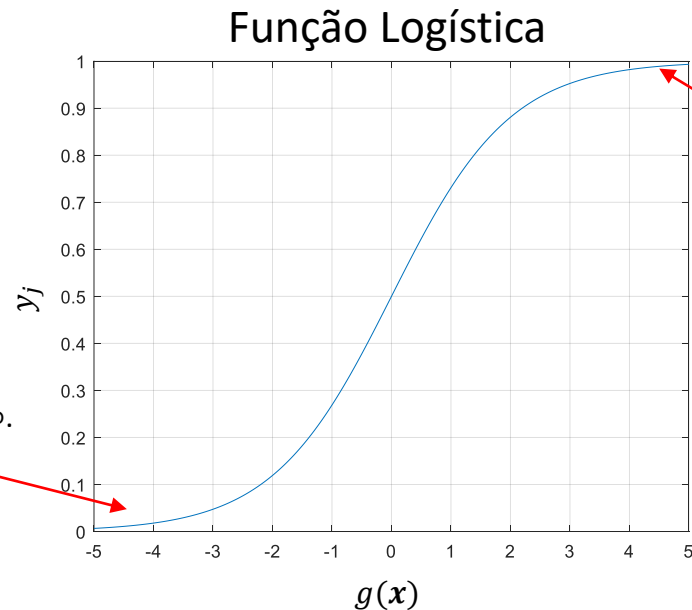
- Sua derivada é dada por

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

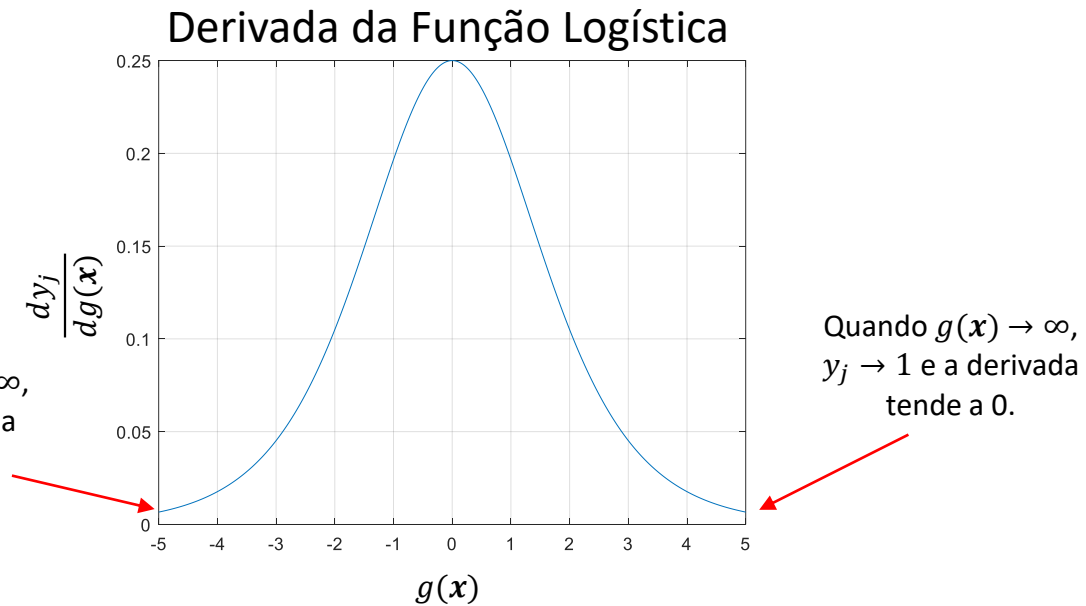
- A derivada será importante durante o processo de aprendizado da rede neural.

# 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  $g(x) = 0$ .**



Quando  $g(x) \rightarrow -\infty$ ,  
 $y_j \rightarrow 0$  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

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

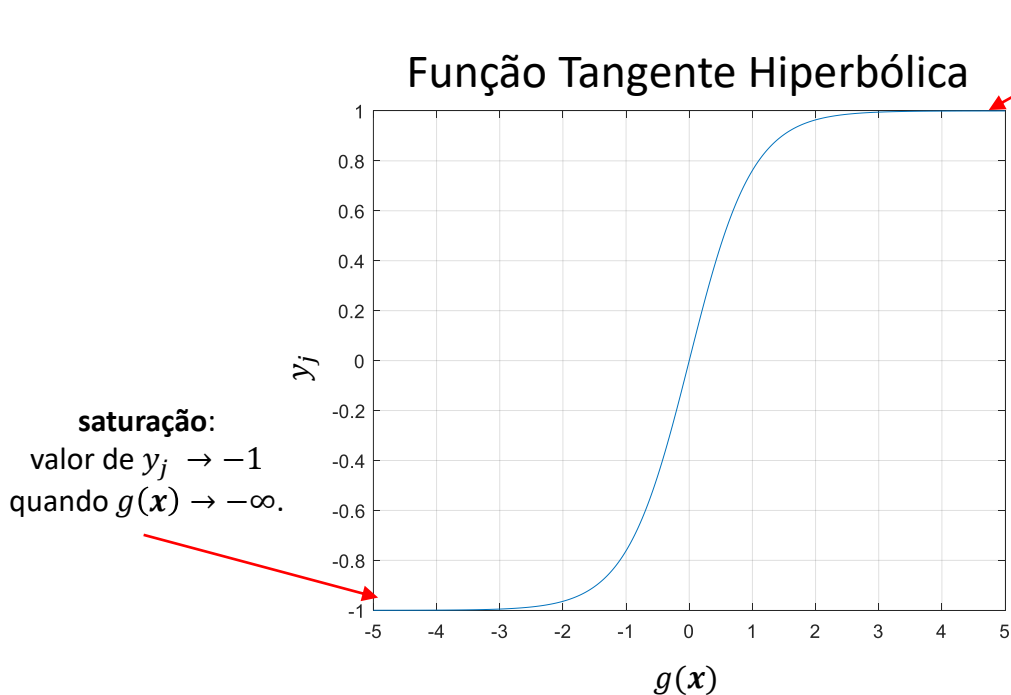
onde  $g(\mathbf{x})$  é a **combinação linear das entradas do nó**.

- Sua derivada é dada por

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

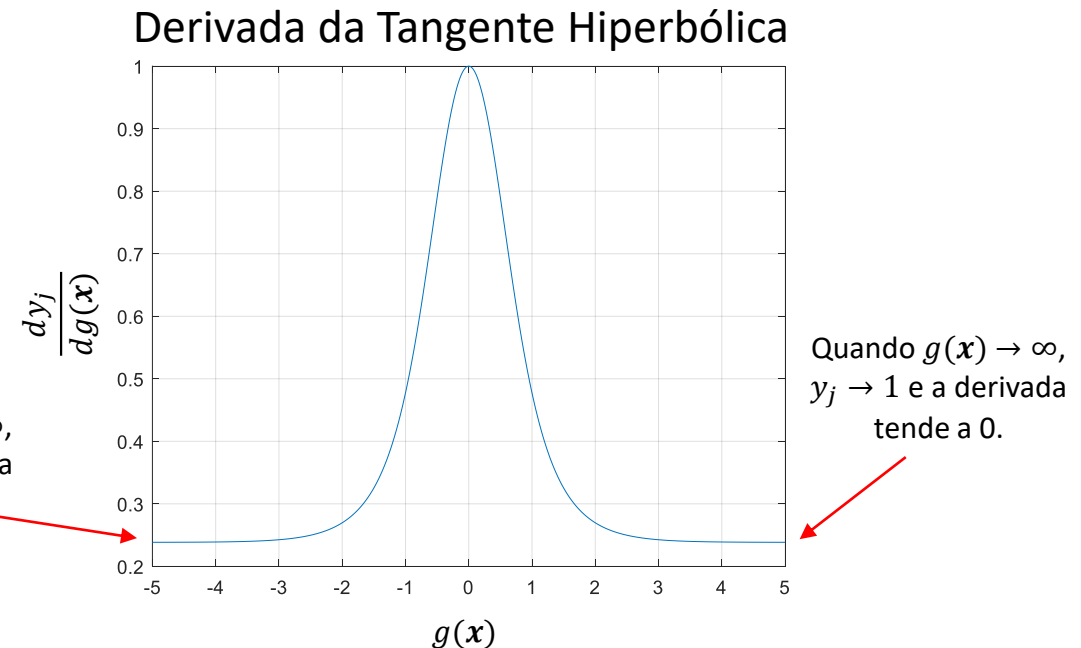
# Função tangente hiperbólica e sua derivada

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



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

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

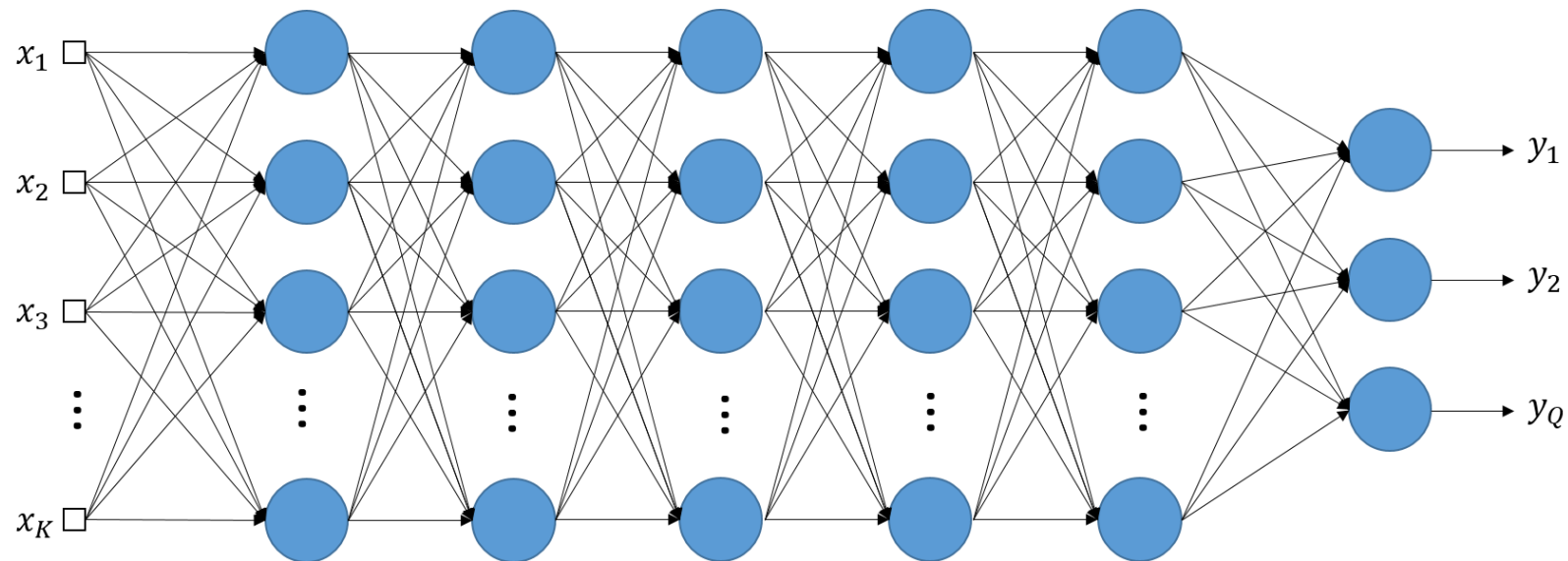


***Na sequência, veremos que esses valores de derivadas menores do que 1 causam um problema no aprendizado de redes com muitas camadas, i.e., redes profundas.***



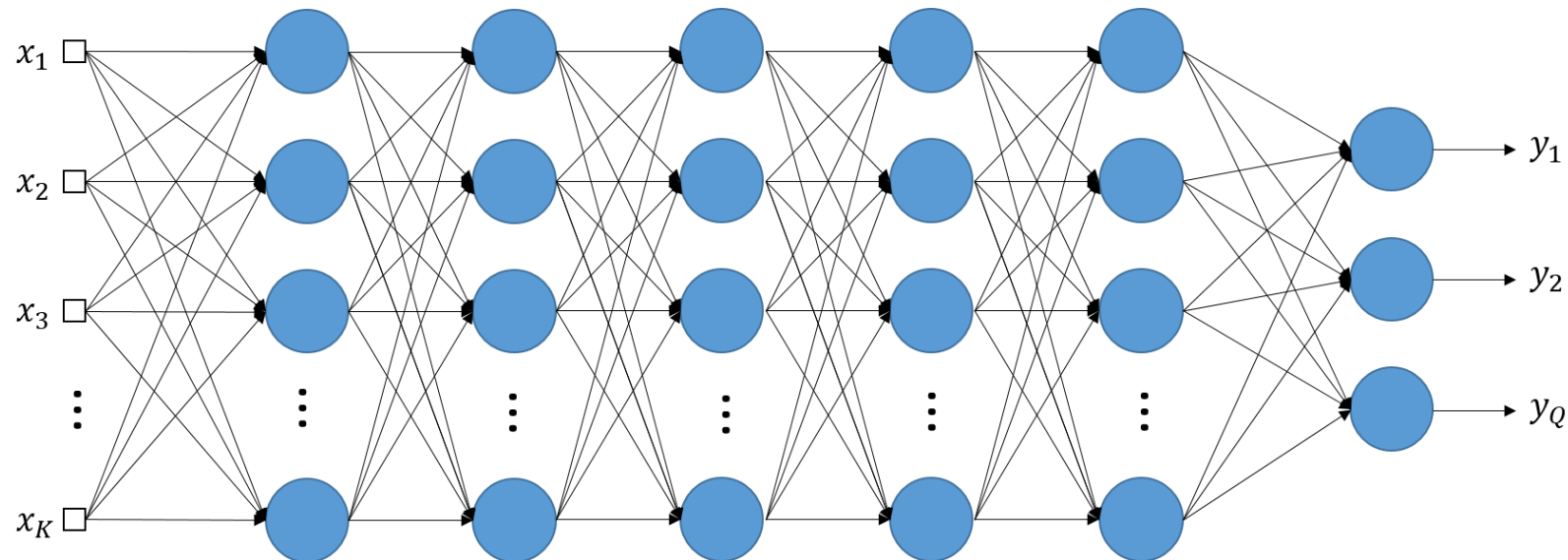
# O problema da dissipação do gradiente

- 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**.



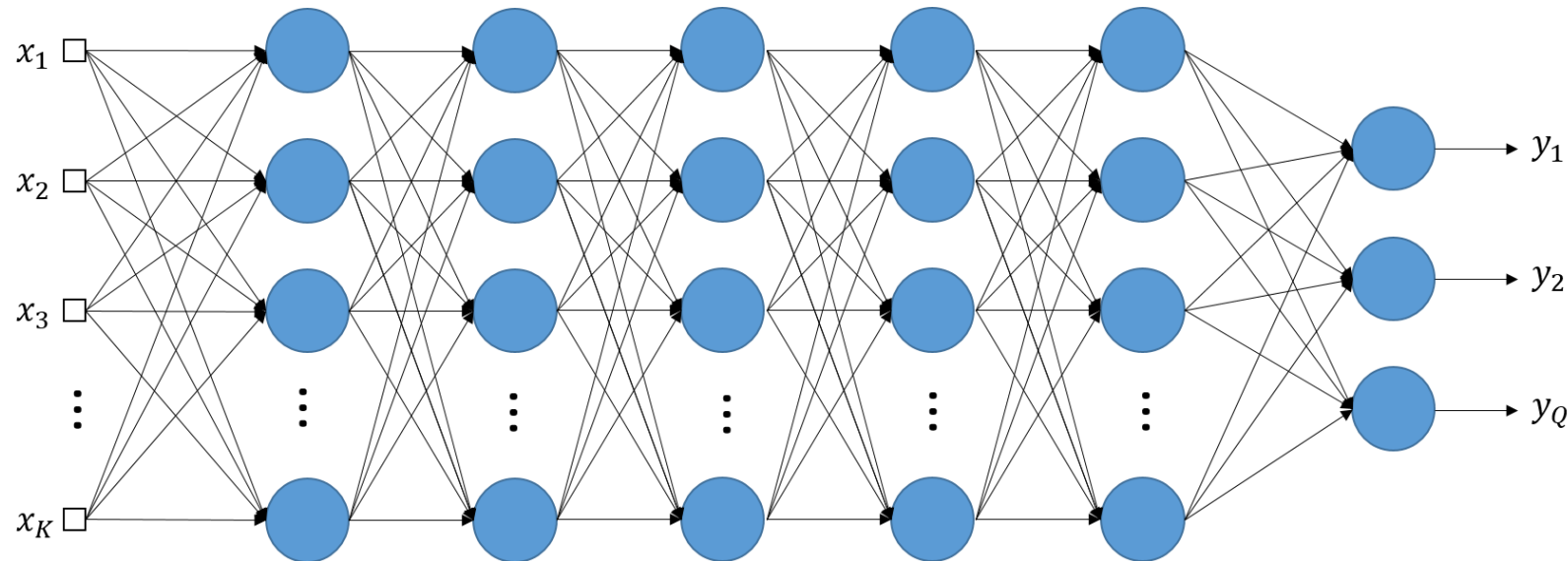
# O problema da dissipação do gradiente

- 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**.



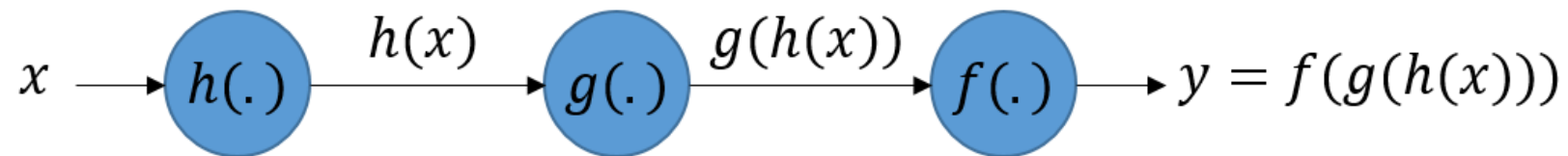
# O problema da dissipação do gradiente

- Em suma, 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  $f(\cdot)$ ,  $g(\cdot)$ , e  $h(\cdot)$  podem ser interpretadas como sendo as funções de ativação dos nós.



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

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

# Regra da cadeia

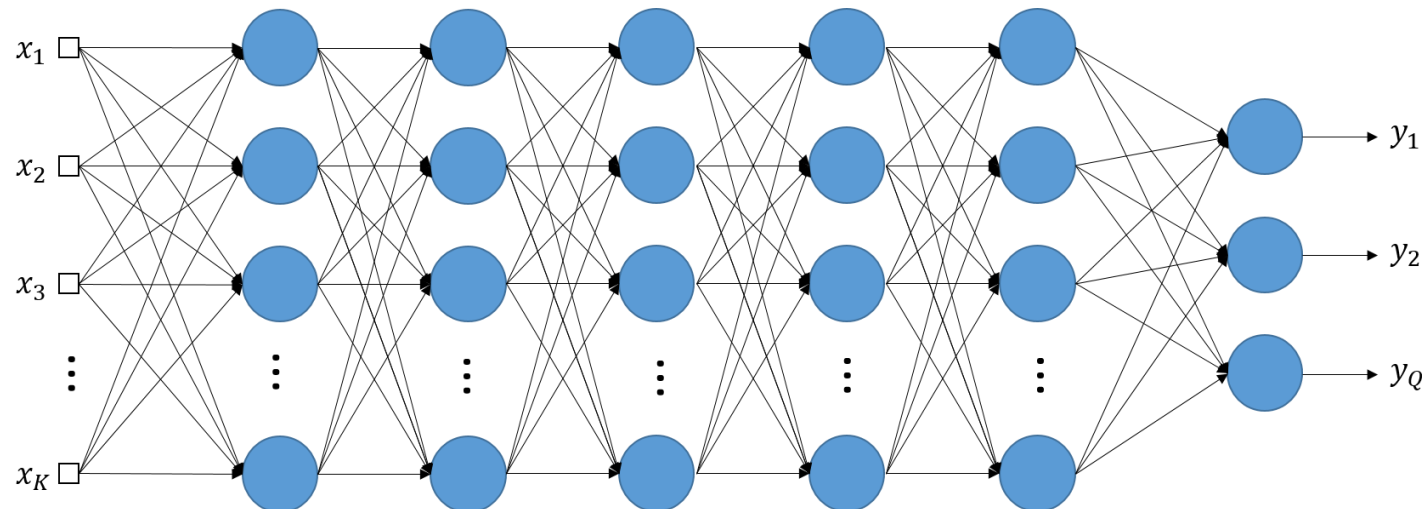
- Em outras palavras, devido à regra da cadeia, o **vetor gradiente** para a **atualização dos pesos de uma dada camada** da rede inclui o **produto das derivadas das funções de ativação dos nós desde a camada de saída até a camada desejada**.

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

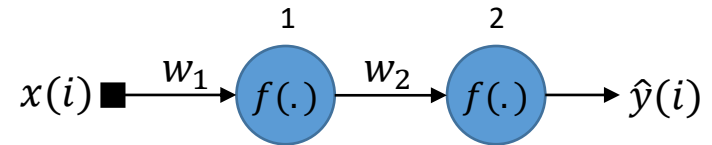
- Lembrem-se que as **funções de ativação**, como **tangente hiperbólica** ou **logística**, têm derivadas no intervalo de 0 até 1.
- Portanto, a multiplicação de vários termos menores do que 1 tende a 0 conforme o número de camadas da rede aumenta.

# O problema da dissipação do gradiente

- Em uma rede com  $M$  camadas, a **retropropagação** tem o efeito de multiplicar até  $M$  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  $M$** .
- 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,  $f(\cdot)$ .
- $g_1 = xw_1 \rightarrow$  entrada (i.e., ativação) do primeiro neurônio.
- $z_1 = f(xw_1) \rightarrow$  saída do primeiro neurônio.
- $g_2 = z_1w_2 = f(xw_1)w_2 \rightarrow$  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

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

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

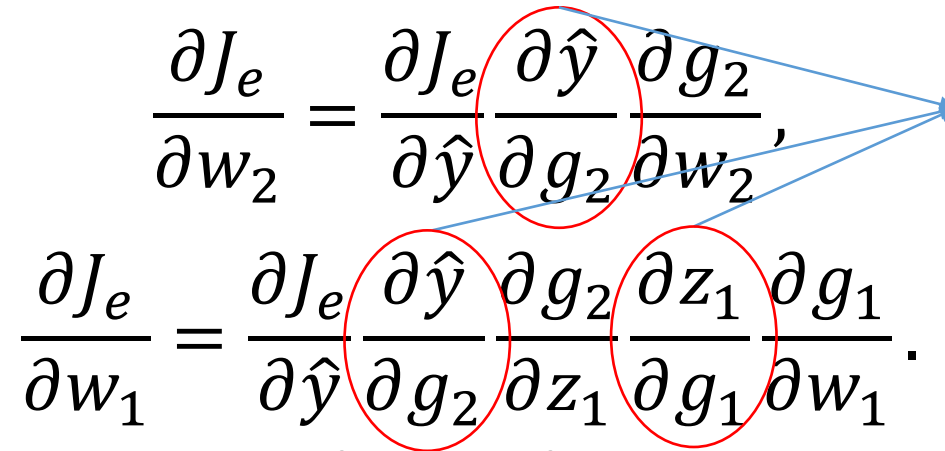
- Usando a regra da cadeia, obtemos as derivadas  $\frac{\partial J_e}{\partial w_1}$  e  $\frac{\partial J_e}{\partial w_2}$

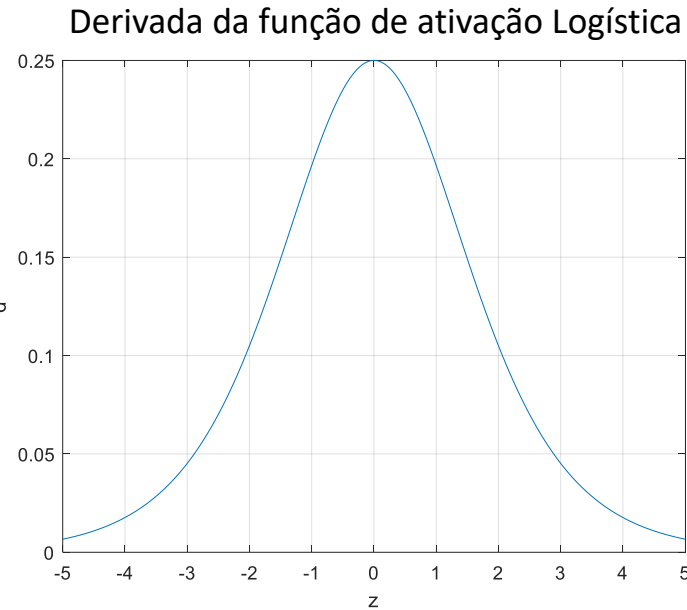
$$\frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2},$$

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



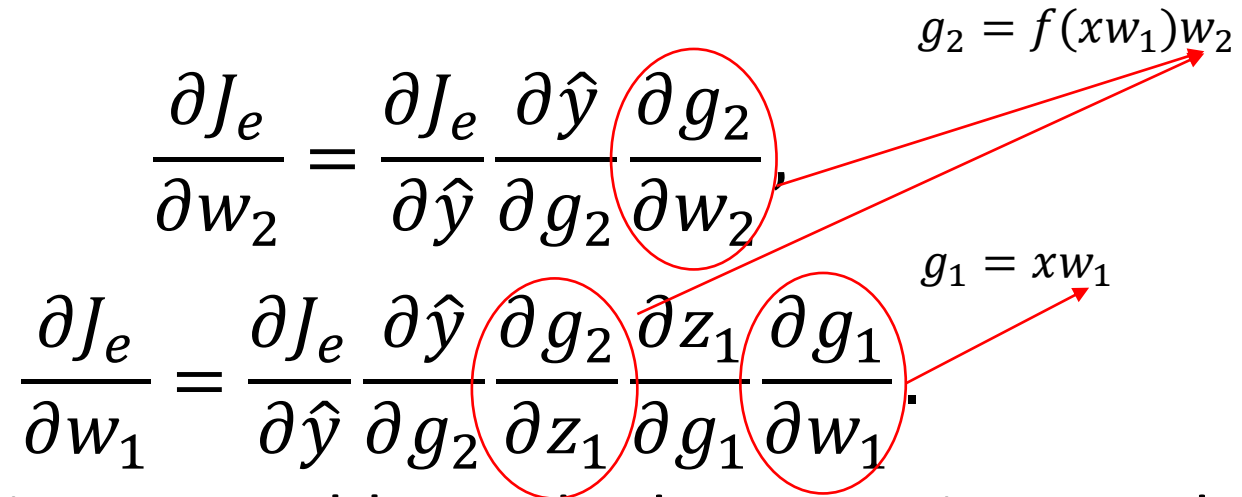
# Dissipação do gradiente

$$\frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2},$$
$$\frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial z_1} \frac{\partial z_1}{\partial g_1} \frac{\partial g_1}{\partial w_1}.$$




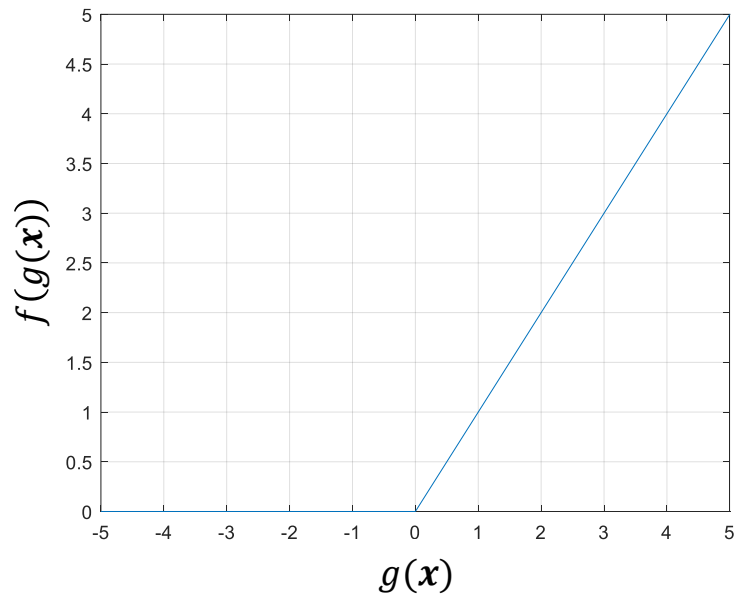
- A derivada da função sigmoide é no máximo igual a 0.25.
- Assim, por exemplo, a primeira camada de uma rede neural com  $M$  camadas, terá as derivadas parciais da função de erro em relação a seus pesos compostas pela multiplicação de  $M$  termos no máximo iguais a 0.25.
- Isso faz com que as primeiras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

# Explosão do gradiente

$$\frac{\partial J_e}{\partial w_2} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial w_2}, \quad g_2 = f(xw_1)w_2$$
$$\frac{\partial J_e}{\partial w_1} = \frac{\partial J_e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial g_2} \frac{\partial g_2}{\partial z_1} \frac{\partial z_1}{\partial g_1} \frac{\partial g_1}{\partial w_1}, \quad g_1 = xw_1$$


- Usando ReLUs, reduzimos o problema do desaparecimento do gradiente.
- Porém, caso os pesos sejam inicializados (aleatório) com valores maiores do que 1, haverá a multiplicação de vários valores assim, resultando 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 ao colapso do treinamento (divergência).

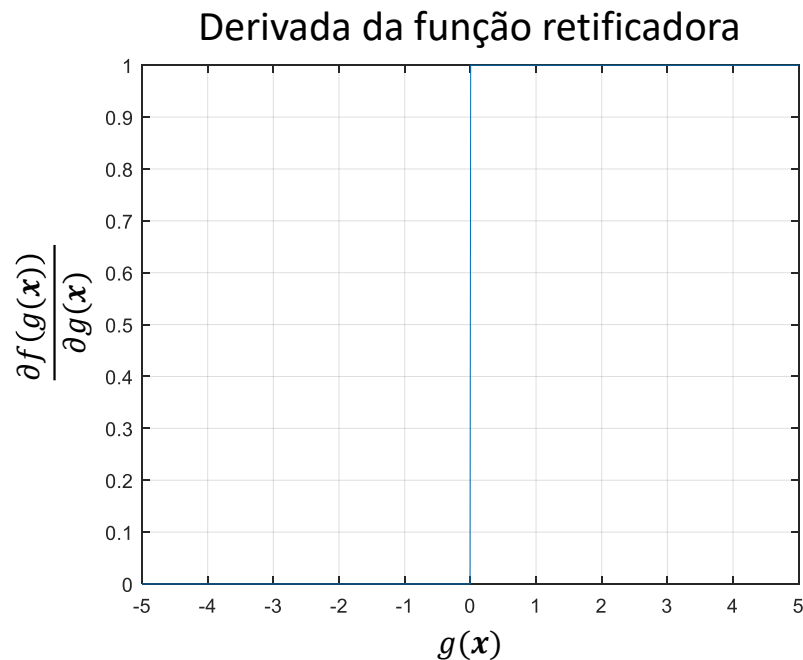
# Função de ativação retificadora



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

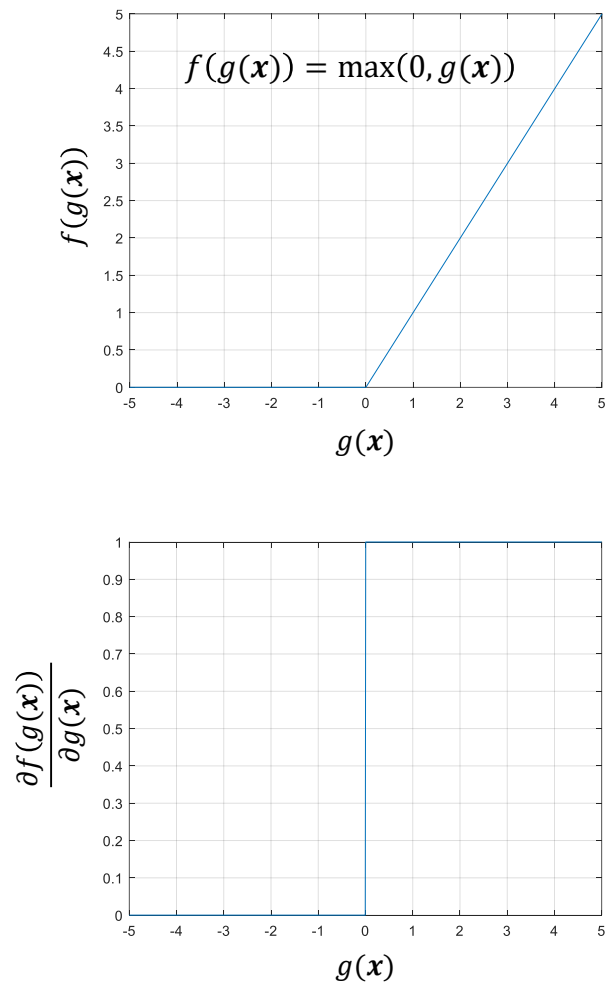
- Com o surgimento das **redes neurais profundas**, e, conseqüentemente, do problema do **desaparecimento do gradiente**, uma outra função de ativação, conhecida como **Rectified Linear Unit (ReLU)**, passou a ser a bastante utilizada.
- É uma **função não-linear** onde sua saída é igual 0 quando  $g(x) \leq 0$  e o próprio  $g(x)$  quando  $g(x) > 0$ .
- 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  $g(x) > 0$ .
- Sua derivada é dada por
$$\frac{dy_j}{dg(x)} = \frac{df(g(x))}{dg(x)} = \begin{cases} 0, & \text{se } g(x) < 0 \\ 1, & \text{se } g(x) > 0 \end{cases}$$
- A derivada é indeterminada para  $g(x) = 0$ .

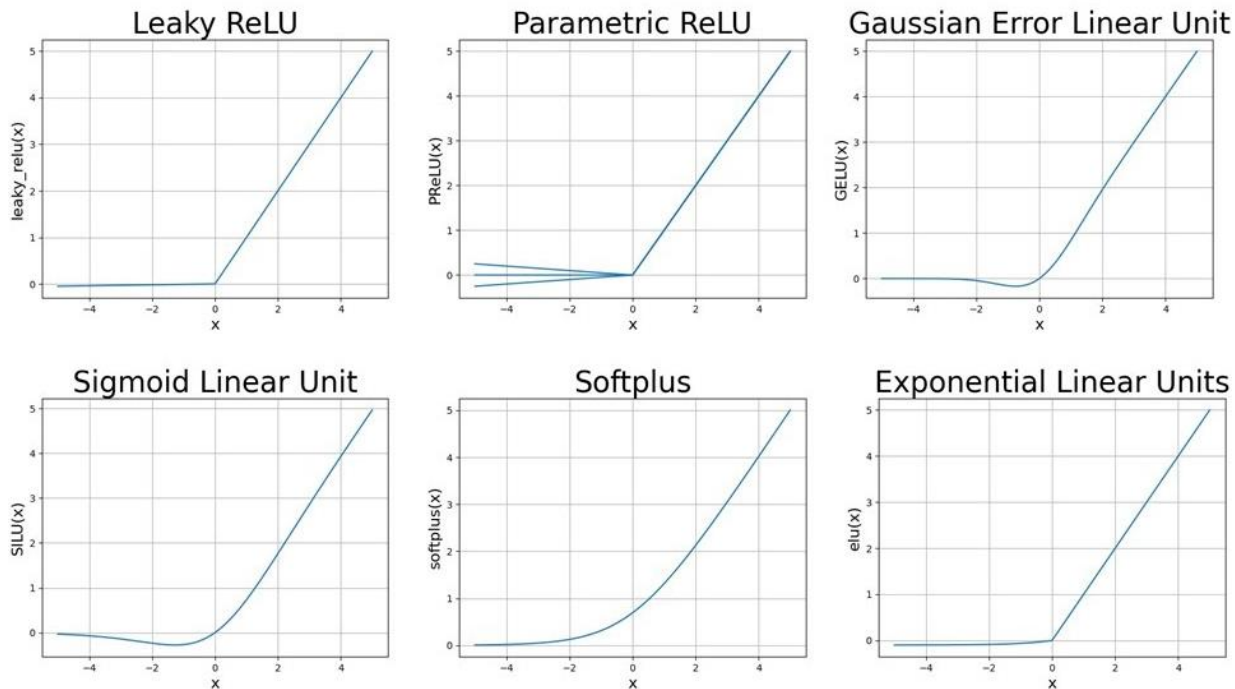
# Função de ativação retificadora



- Uma desvantagem é que ela causar o problema conhecido como **ReLU agonizante**.
- Esse **problema ocorre durante o treinamento** da rede, quando a ativação do nó,  $g(x)$ , 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**.

# Função de ativação retificadora

## ReLU Variants



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

# Como minimizar a dissipação e explosão do gradiente?

- Algumas formas de se minimizar esses problemas são:
  - **Inicialização apropriada dos pesos:** *garantem que as variâncias de saída das camadas (e de seus respectivos gradientes) não aumentem ao longo da rede*, ajudando a mitigar ambos os problemas.
  - **Normalização (padronização) de batch:** *mantém as ativações de cada camada da rede próximas à média zero e desvio padrão unitário* durante o treinamento, minimizando ambos os problemas.
  - **Poda do gradiente:** *limita (poda) os valores dos gradientes* durante o treinamento para que eles excedam algum limite pré-definido, mitigando apenas o problema da explosão do gradiente.

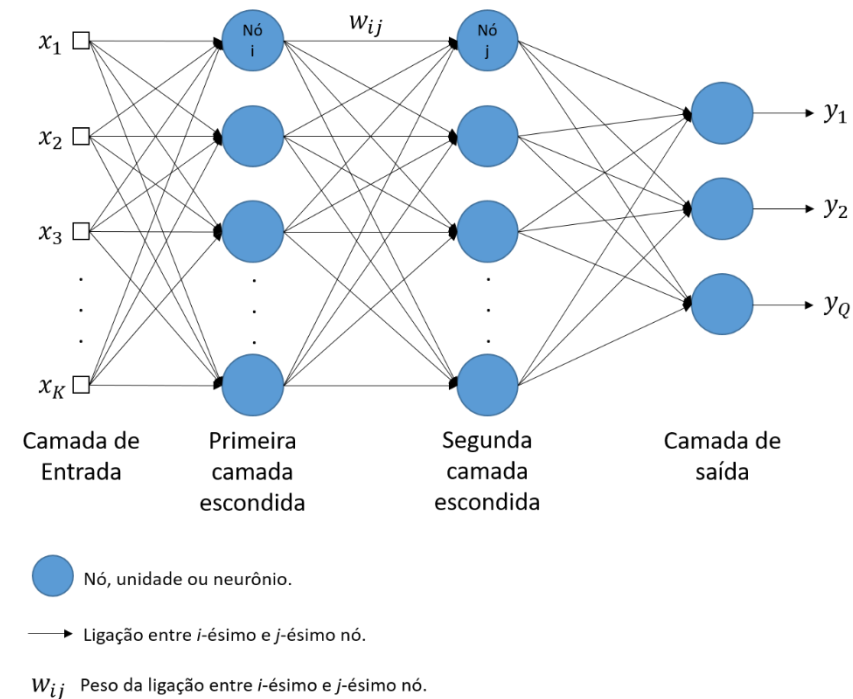
# Tarefa

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

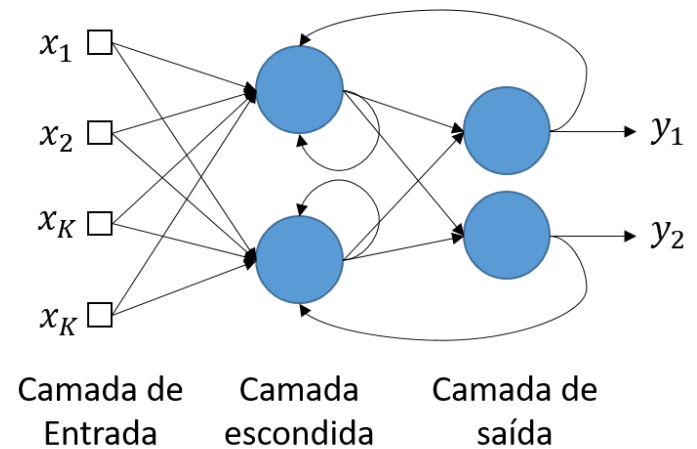


# Conectando Neurônios

- 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.

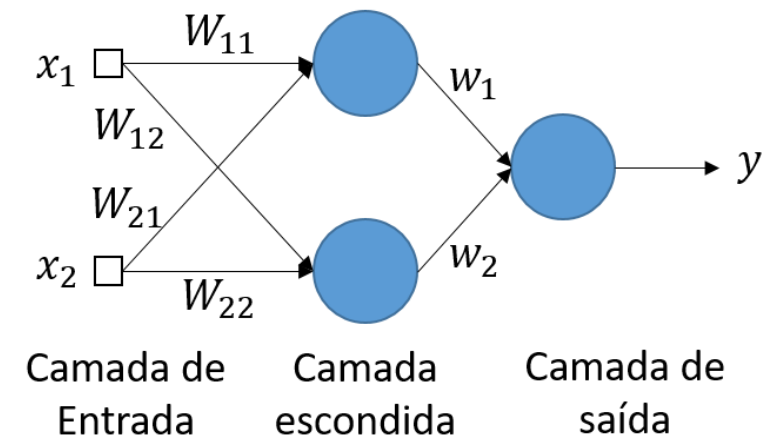


# Conectando Neurônios



- 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

$$y = f(\mathbf{w}^T f(\mathbf{W}^T \mathbf{x})),$$

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

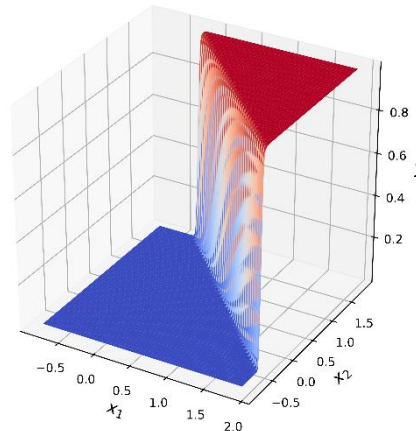
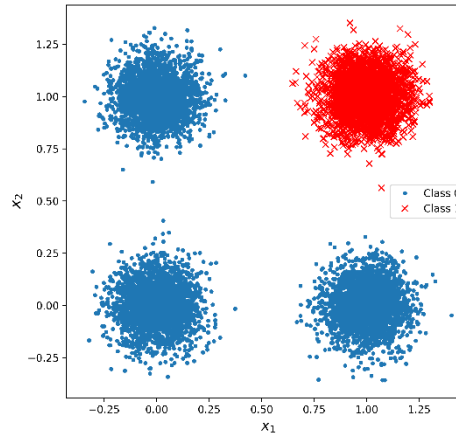
- Percebam que a saída da rede é dada pelo **aninhamento** das saídas de **funções de ativação não-lineares**.
- Sendo assim, as funções que uma rede neural pode representar (i.e., aproximar) podem ser **altamente não-lineares** dependendo da quantidade de camadas e nós.
- Portanto, redes neurais podem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

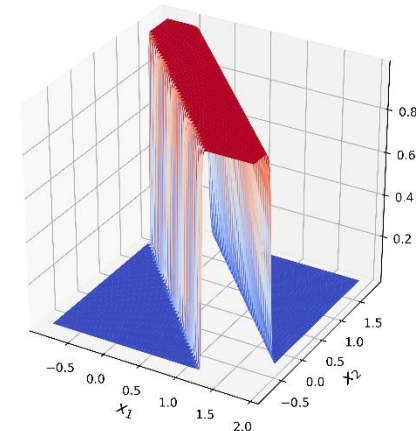
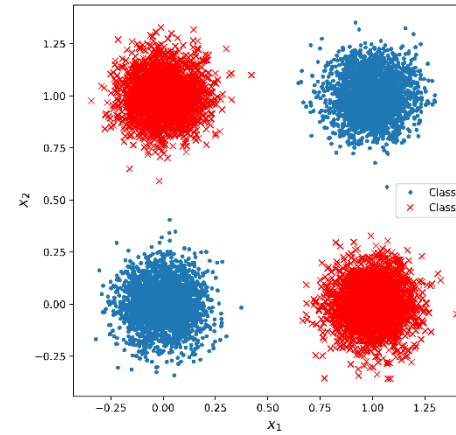
[Exemplo: FunctionApproximationWithMLP.ipynb](#)

- 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.

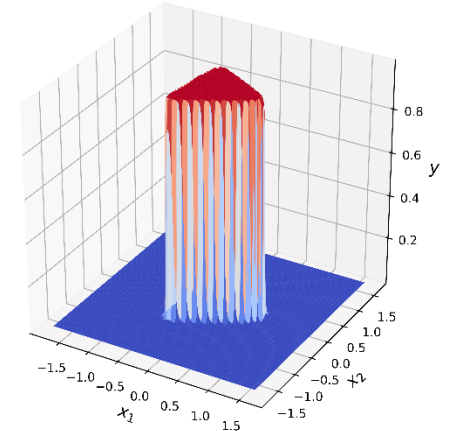
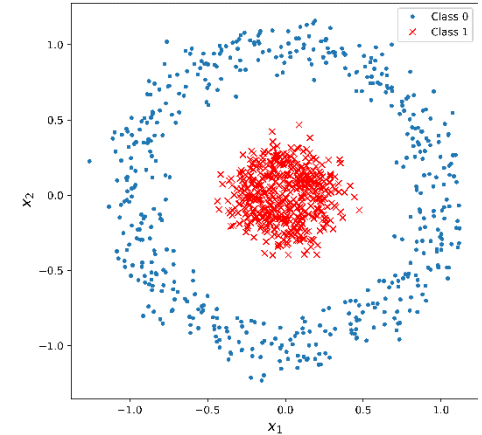
Função AND: MLP com 0 camadas escondidas, apenas um neurônio na camada de saída.  
Total: 1 nó.



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.

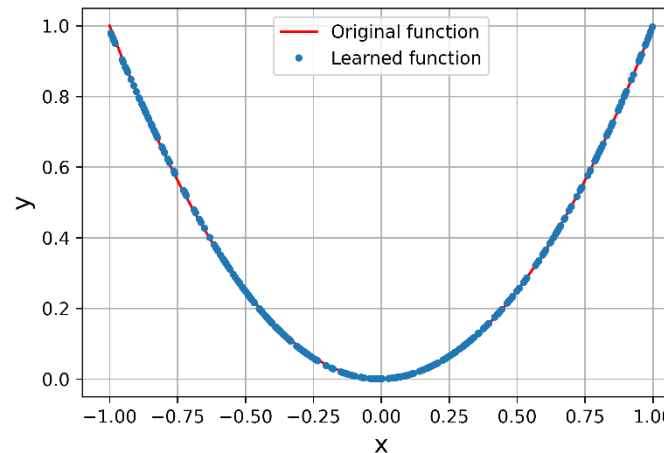


# Aproximação universal de funções: Regressão

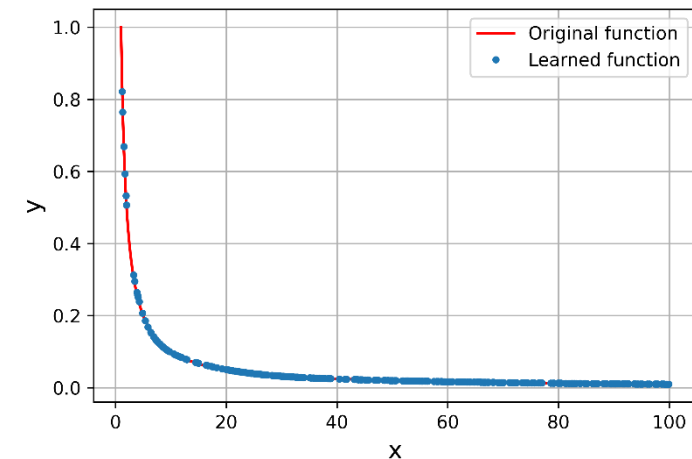
- Redes neurais podem ser usadas para aproximar funções como as mostradas abaixo:

- $f(x) = x^2, -1 \leq x \leq 1,$
- $f(x) = \frac{1}{x}, 1 \leq x \leq 100,$
- $f(x) = \sin(x), 1 \leq x \leq 2\pi.$

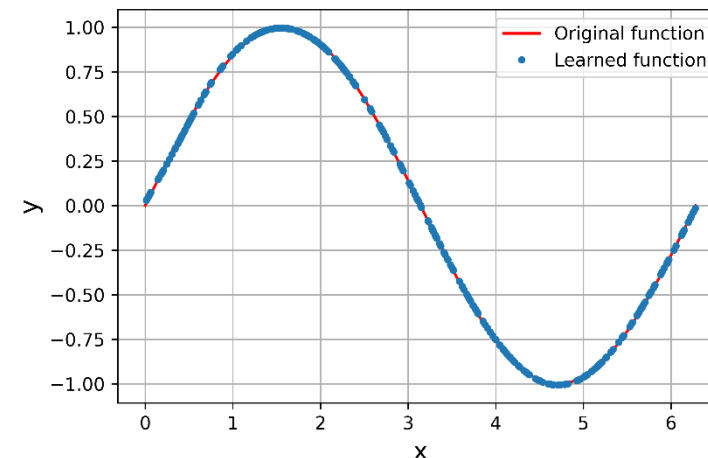
$$f(x) = x^2$$



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



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



# 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..



## 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

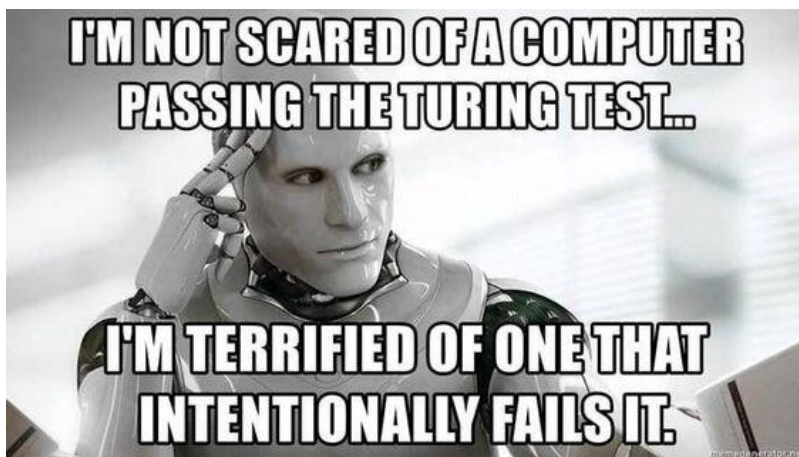


What I think I do

```
In [1]:  
import keras  
Using TensorFlow backend.
```

What I actually do

I'M NOT SCARED OF A COMPUTER  
PASSING THE TURING TEST...



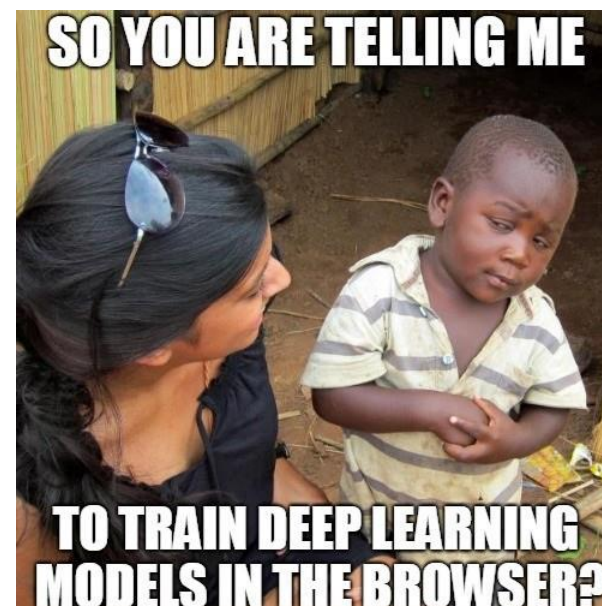
I'M TERRIFIED OF ONE THAT  
INTENTIONALLY FAILS IT.

Dog



I NEED GPU  
FOR MY DUMB  
NEURAL NETWORK

SO YOU ARE TELLING ME



TO TRAIN DEEP LEARNING  
MODELS IN THE BROWSER?

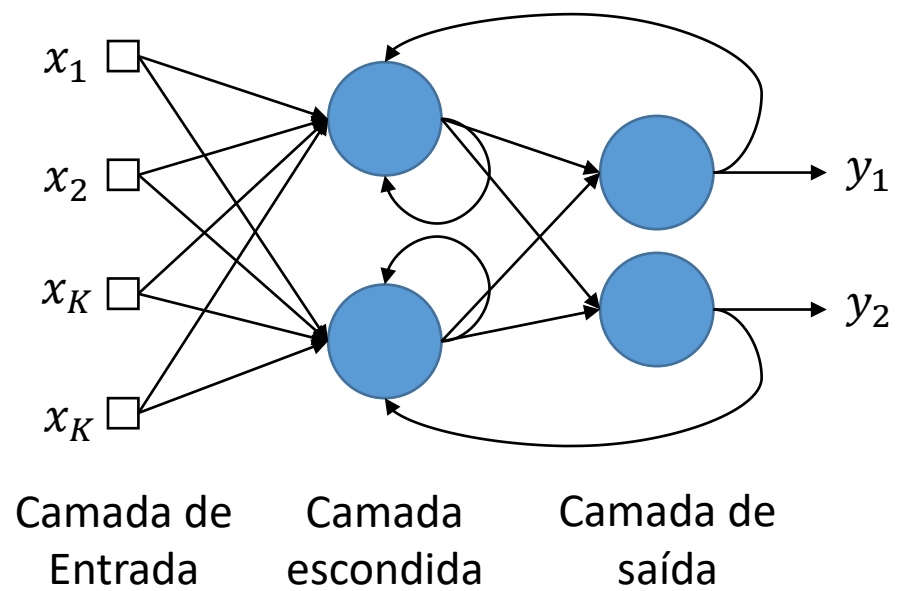
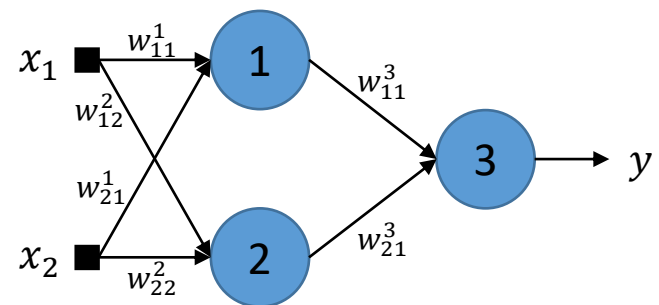
ONE DOES NOT SIMPLY



GENERATE MEMES USING DEEP  
LEARNING



Figuras



# 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  $h(\cdot)$ ,  $g(\cdot)$  e  $f(\cdot)$ .
  - 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

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

