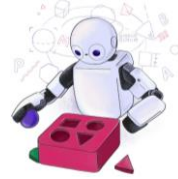


# TP555 - Inteligência Artificial e Machine Learning: *Redes Neurais Artificiais (Parte I)*



**Inatel**

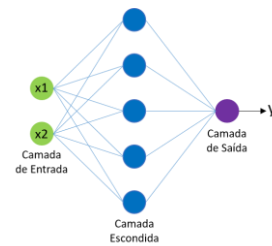
Felipe Augusto Pereira de Figueiredo  
felipe.figueiredo@inatel.br

Agora vamos falar sobre um tópico que parece, inicialmente, não ser relacionado: o cérebro. Entretanto, como veremos à seguir, as idéias que discutimos até agora são úteis na construção de modelos matemáticos da atividade do cérebro. E como veremos, essas ideias que já discutimos, nos ajudaram a entender o funcionamento das redes neurais artificiais.

Redes neurais artificiais são um das formas mais populares e efetivas para implementação de sistemas de aprendizado e merecem por si só uma disciplina em separado.

## Redes Neurais Artificiais

- Redes neurais artificiais (RNAs) são modelos computacionais inspirados pelo funcionamento do cérebro dos animais.
- Elas são capazes de realizar aprendizado de máquina bem como o reconhecimento de padrões.
- RNAs são geralmente apresentadas como sistemas de **neurônios interconectados**, que podem computar valores de saída, simulando o comportamento de redes neurais biológicas.
- Esta primeira parte da aula foca nos elementos básicos de uma rede neural, os **neurônios**.



## Algumas aplicações famosas

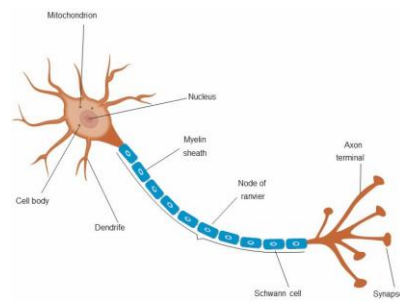
- RNAs são versáteis, poderosas e escalonáveis, tornando-as ideais para realizar tarefas grandes e altamente complexas de **aprendizado de máquina**, como por exemplo classificar bilhões de imagens (por exemplo, Google Images), serviços de reconhecimento de fala (por exemplo, o Siri da Apple, Alexa da Amazon e Google Assistant da Google), recomendar os melhores vídeos a centenas de milhões de usuários todos os dias (por exemplo, YouTube) ou aprender a vencer o campeão mundial de Go examinando milhões de jogos anteriores e depois jogando contra si mesmo (AlphaGo do DeepMind).



RNAs são versáteis, poderosas e escalonáveis, tornando-as ideais para realizar tarefas grandes e altamente complexas de Machine Learning, como classificar bilhões de imagens (por exemplo, Google Images), ativar serviços de reconhecimento de fala (por exemplo, o Siri da Apple), recomendar os melhores vídeos assistir a centenas de milhões de usuários todos os dias (por exemplo, YouTube) ou aprender a vencer o campeão mundial no jogo Go examinando milhões de jogos anteriores e depois jogando contra si mesmo (AlphaGo do DeepMind).

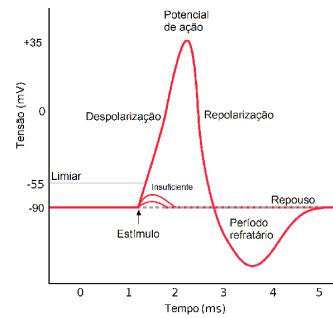
## Um pouco de contexto

- A descoberta da célula em 1665 por Robert Hooke foi importantíssima para que houvesse uma melhor compreensão da estrutura dos seres vivos.
- Podemos considerar a célula como sendo o “**átomo da vida**”.
- As células **eucariontes** possuem três partes principais: membrana, núcleo e citoplasma. A membrana “delimita a célula”, i.e., ela isola seu interior do meio externo. Já o núcleo abriga o material genético e, no citoplasma, estão componentes como as organelas.
- **Neurônios** são células também, mas são células que possuem mecanismos elétricos e/ou químicos característicos. A figura ao lado mostra o diagrama de um **neurônio**.



## Um pouco de contexto

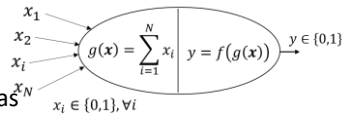
- Em termos simples, mas lembrando de que há exceções, nós podemos afirmar que:
  - O neurônio recebe estímulos elétricos, basicamente a partir dos dendritos.
  - Esses estímulos são integrados.
  - A integração dos estímulos pode levar à geração ou não de uma resposta elétrica enviada pelo axônio.
- Do ponto de vista do nosso curso, o **neurônio** será um sistema com várias entradas e uma saída.
- Nós podemos simplificar o funcionamento do **neurônio** como:
  - Os neurônios recebem estímulos elétricos.
  - Esses estímulos são integrados.
  - Se a atividade (i.e., integração dos estímulos) exceder certo limiar, o **neurônio** gera um pulso (ou potencial de ação).
- O potencial de ação é mostrado na figura ao lado.
- Um **neurônio** se conecta com 10 a 100000 outros **neurônios** através das **sinapses**.
- Sinais são passados de **neurônio** para **neurônio** através de reações eletro-químicas.



## O Modelo de McCulloch e Pitts

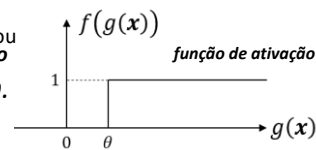
- A figura ao lado mostra o modelo matemático do **neurônio** criado por McCulloch e Pitts em 1943.
- A grosso modo, o **neurônio** é ativado (ou disparado) quando uma **combinação linear** de suas entradas excede um **limiar de ativação**.
- Ou seja, um **neurônio** nada mais é do que um **classificador linear** que vimos anteriormente.
- As premissas do modelo do **neurônio** de McCulloch e Pitts (M-P) são:

- Os valores das entradas,  $x_i, \forall i$ , ou também chamadas de **sinapses**, são sempre valores booleanos, i.e., '0', ou '1'.
- As entradas são simplesmente somadas.
- A atividade do **neurônio** é um processo do tipo "tudo ou nada", ou seja, um processo binário. Portanto, a **função de ativação** do neurônio é uma **função degrau** com **ponto de disparo** dependente do **limiar de ativação**,  $\theta$ .
- Um certo número de **sinapses** deve ser excitado num determinado período para que o neurônio "dispare".



$$y = f(g(x)) = \begin{cases} 1, & \text{se } g(x) \geq \theta \\ 0, & \text{se } g(x) < \theta \end{cases}$$

onde  $\theta$  é o **limiar de ativação**.



O final do século XIX e o início do século XX foram períodos fundamentais para o estabelecimento da visão atual do sistema nervoso. Quando Warren McCulloch e Walter Pitts apresentaram o primeiro modelo "computacional" de neurônio em 1943, já haviam sido identificados vários elementos considerados centrais até hoje.

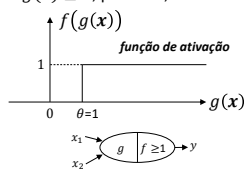
O modelo de McCulloch e Pitts parece "simples" quando comparado à alguns modelos empregados hoje em dia, mas sua importância foi enorme. A partir desse modelo, foi possível estabelecer uma conexão entre o funcionamento de um neurônio e a lógica proposicional. A partir daí, a relação com a computação digital foi natural.

Artigo seminal de McCulloch e Pitts: "A Logical Calculus of Ideas Immanent in Nervous Activity," W. McCulloch and W. Pitts (1943)

## Exemplos com o neurônio de McCulloch e Pitts

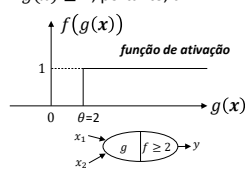
OR			
x1	x2	y	$g(x)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	2

- Qual seria o valor do **limiar de ativação**,  $\theta$ ?
- Analisando-se  $g(x)$ , vemos que o disparo deve ocorrer quando  $g(x) \geq 1$ , portanto,  $\theta = 1$ .



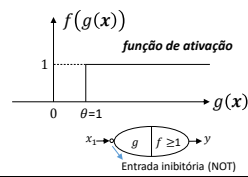
AND			
x1	x2	y	$g(x)$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	2

- Qual seria o valor do **limiar de ativação**,  $\theta$ ?
- Analisando-se  $g(x)$ , vemos que o disparo deve ocorrer quando  $g(x) \geq 2$ , portanto,  $\theta = 2$ .



NOT		
x1	y	$g(x)$
0	1	0
1	0	1

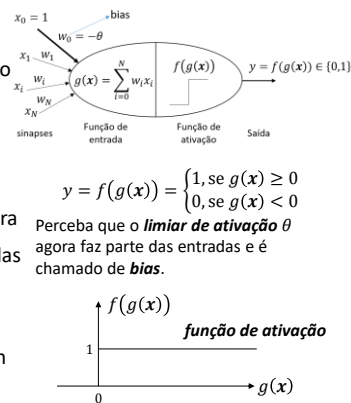
- Qual seria o valor do **limiar de ativação**,  $\theta$ ?
- Analisando-se  $g(x)$ , vemos que para o disparo ocorrer, o valor de  $x_1$  deve ser negado, e assim, ele ocorre quando  $g(x) \geq 1$ , portanto,  $\theta = 1$ .



Entradas inibitórias são entradas que tem seus valores '**negados**'.

## Perceptron

- Em 1958, Frank Rosenblatt, propôs o modelo clássico do **perceptron**.
- Em 1969, o modelo de Rosenblatt foi refinado e cuidadosamente analisado por Minsky e Papert. O modelo criado por eles é chamado de **perceptron**. O modelo proposto por eles é mostrado na figura ao lado.
- O modelo **perceptron**, é um modelo computacional mais geral que o modelo do **neurônio** de M-P.
- Esse novo modelo supera algumas das limitações do modelo de M-P, introduzindo o conceito de **pesos sinápticos** (uma medida de importância dos atributos) para as entradas (ou **sinapses**) e um método para aprender esses **pesos**. Além disso, as entradas não são mais limitadas a valores booleanos, como no caso do modelo de M-P, suportando entradas reais, o que torna este modelo mais útil e generalizado.
- Assim como no modelo de M-P, a **função de ativação** utilizada pelo **perceptron** também é a **função degrau** com a diferença que aqui ela não mais depende do **limiar de ativação**  $\theta$ .



Nós podemos dizer que o perceptron é uma rede neural com um único neurônio. O Perceptron é um modelo computacional mais geral que o neurônio de Mcculloch Pitts.

A principal diferença entre o neurônio de Mcculloch Pitts e o Perceptron é uma introdução de pesos sinápticos para entradas e um método para aprender os valores desses pesos.

Relembrando como o modelo de M-P é ativado (i.e., disparado):

$$y = f(g) \quad g(x) = \sum_{i=1}^N w_i x_i \geq \theta$$

Reescrevendo  $g(x) = \sum_{i=1}^N w_i x_i$

$$y = f(g) \quad g(x) = \sum_{i=1}^N w_i x_i \geq \theta \quad \Rightarrow \quad \sum_{i=1}^N w_i x_i - \theta \geq 0$$

Passando  $\theta$  para o lado esquerdo da desigualdade, tem-se

$$y = f(g) \quad g(x) = \sum_{i=1}^N w_i x_i - \theta \geq 0$$

Se considerarmos  $x_0 = 1$  e  $w_0 = -\theta$ , então podemos reescrever a equação acima como

$$y = f(g) \quad g(x) = \sum_{i=0}^N w_i x_i \geq 0$$

Desta forma, o limiar de ativação é absorvido pelo somatório e podemos usar a função de ativação centrada em zero, pois agora, ajusta-se o limiar de ativação indiretamente, através da atualização do peso  $w_0$ , que é normalmente chamado de **bias**.

A partir da equação acima, deve ficar claro que mesmo um Perceptron separa o espaço de entrada em duas metades. Todas as entradas que produzem 1 ficam em um lado e todas as entradas que produzem 0 ficam em outro lado.



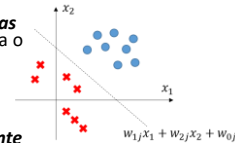


## Perceptron

- A ideia é que a ativação do **perceptron** (causada pelos estímulos de entrada) seja uma **combinação linear** entre os **estímulos** e os **pesos sinápticos**. Se essa ativação exceder certo **limiar de ativação**, ocorrerá o **disparo**. Isso pode ser expresso por meio de uma **função de ativação** do tipo **degrau**.
- Note que a **função de ativação**  $f(\cdot)$  está centrada “em torno de zero” e o **limiar de ativação** (ou **disparo**) é controlado, indiretamente, pelo valor do **peso do bias**,  $w_0$ .
- O tipo de resposta do **perceptron** dá origem a um **classificador** para **problemas com duas classes**. As classes são separadas por uma **fronteira de decisão** para a qual a equação abaixo é verdadeira.

$$\sum_{i=0}^N w_i x_i = 0.$$

- No **espaço dos atributos**  $x_i, \forall i$ , essa é a equação de um **hiperplano**.
- Portanto, um **perceptron** só é capaz de **classificar** dados que sejam **linearmente separáveis** (ou seja, separáveis por um **hiperplano**).
- O **perceptron** convergirá apenas se o conjunto de dados for **linearmente separável**. A figura ao lado ilustra isso para um caso bidimensional.
- Observe que, ao contrário dos **classificadores de regressão logística**, os **perceptrons** não produzem como saída uma probabilidade de classe, em vez disso, eles apenas fazem previsões com base em um **limiar rígido**, i.e., 0 ou 1. Essa é uma das razões para se preferir a **regressão logística** ao invés do **perceptron**.



Existem várias outras funções de ativação:

[https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)

## Regra de aprendizado do perceptron

- Como discutimos anteriormente, a **função degrau** tem derivada igual a 0 em todos os pontos, exceto em torno de 0, onde ela é indefinida. Portanto, nós não podemos utilizar o **gradiente descendente** para treinar o **perceptron**.
- Existe, porém, uma regra simples de atualização dos **pesos** que converge para uma solução, ou seja, um **separador linear** que **classifica** os dados perfeitamente, dado que eles sejam **linearmente separáveis**.
- Portanto, caso os dados sejam **linearmente separáveis**, a **regra de aprendizado do perceptron** tem convergência garantida num número finito de iterações. Nessa regra, para cada exemplo do conjunto de treinamento, obtém-se, primeiramente, a saída do **perceptron** para os **pesos sinápticos** atuais:

$$y = f\left(\sum_{i=0}^N w_i x_i\right) = f(\mathbf{w}^T \mathbf{x}).$$

- Em seguida, calcula-se o erro entre a saída  $y$  do **perceptron** e o rótulo  $d$  do exemplo:

$$e = d - y.$$

- Caso o erro não seja nulo, a **equação de adaptação dos pesos sinápticos** é definida da seguinte forma:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha e \mathbf{x},$$

onde  $\alpha$  é a **taxa** (ou **passo**) de aprendizagem.

- Após a apresentação de todos os exemplos de treinamento (ou seja, uma **época**), deve haver um **embaralhamento** dos exemplos e uma nova etapa de treinamento. No caso ótimo, quando a **separação linear** ocorrer, não haverá mais erros, e as **regras de atualização** calculadas não mais modificarão os **pesos sinápticos**.
- **OBS.:** A **regra de aprendizado do perceptron** é, geralmente, aplicada a um exemplo de entrada por vez. Os exemplos são escolhidos aleatoriamente, assim como o que é feito com o **gradiente descendente estocástico**.

Perceba que a **equação de adaptação dos pesos sinápticos** é idêntica à equação de atualização que encontramos para regressores lineares.

Como ambos, o rótulo  $d$  e o valor de saída do perceptron  $y$ , assumem apenas 2 valores, 0 ou 1, existem apenas 3 possibilidades para a equação de atualização dos pesos:

- Se a saída for correta, i.e.,  $d=y$ , então os pesos não são atualizados.
- Se  $d = 1$  mas  $y=0$ , então o valor do peso é aumentado caso a entrada correspondente,  $x_i$ , seja positiva e diminuído caso  $x_i$  seja negativo. Isso faz sentido pois nós queremos que o valor de  $\mathbf{w}^T \mathbf{x}$  aumente tal que  $y$  se torne 1.
- Se  $d = 0$  mas  $y=1$ , então o valor do peso é diminuído caso a entrada correspondente,  $x_i$ , seja positiva e aumentado caso  $x_i$  seja negativo. Isso faz sentido pois nós queremos que o valor de  $\mathbf{w}^T \mathbf{x}$  diminua tal que  $y$  se torne 0.

## Exemplo: Perceptron com SciKit-Learn

```
import numpy as np
from sklearn.linear_model import Perceptron
from sklearn.metrics import mean_squared_error

# Define the number of examples.
N = 1000

# Create dataset.
x1 = np.random.randint(0,2,N)
x2 = np.random.randint(0,2,N)

y = x1 & x2

x1 = x1 + 0.1*np.random.randn(N)
x2 = x2 + 0.1*np.random.randn(N)

x0 = np.ones((N,))
X = np.c_[x0,x1,x2]

# Instantiate and train perceptron.
per = Perceptron(random_state=42)
per.fit(X, y)

# Predict.
y_pred = per.predict(X)

# Calculate MSE.
error = mean_squared_error(y_pred, y)
```

Importa classe Perceptron.

Gera os rótulos à partir dos dados originais. Função lógica AND.

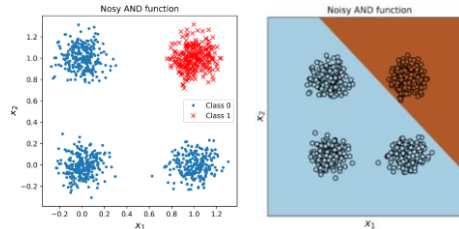
Adiciona ruído aos atributos de entrada

Cria vetor de 1s para o peso de bias.

Instancia e treina o Perceptron.

Realiza a predição.

Calcula erro quadrático médio.



- Exemplo de classificação de dados ruidosos linearmente separáveis.
- A base de dados é gerada à partir da função de uma porta lógica AND.
- Como podemos ver, o perceptron classifica perfeitamente o conjunto de dados ruidosos.


Como você pode ter percebido, o algoritmo de aprendizado do Perceptron se parece muito com o do gradiente descendente estocástico. De fato, a classe Perceptron da biblioteca Scikit-Learn é equivalente a usar um SGDClassifier com os seguintes hiperparâmetros: `loss = "perceptron"`, `learning_rate = "constant"`, `eta0 = 1` (a taxa de aprendizado) e `penalty = None` (sem regularização).

## Avisos

- Material já se encontra no site.

Obrigado!


THE #1 deep learning expert EXCUSE FOR LEGITIMATELY SLACKING OFF:  
"MY model's training"



HEY! GET BACK TO WORK!

training!

OH. CARRY ON.



STILL WAITING  
FOR MY NEURAL NETWORK TO TRAIN


OH. HEY, YOU ORGANIZED OUR PHOTO ARCHIVE!

YEAH, I TRAINED A NEURAL NET TO SORT THE UNLABELED PHOTOS INTO CATEGORIES.

WHOA! NICE WORK!

ENGINEERING TIP:  
WHEN YOU DO A TASK BY HAND,  
YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

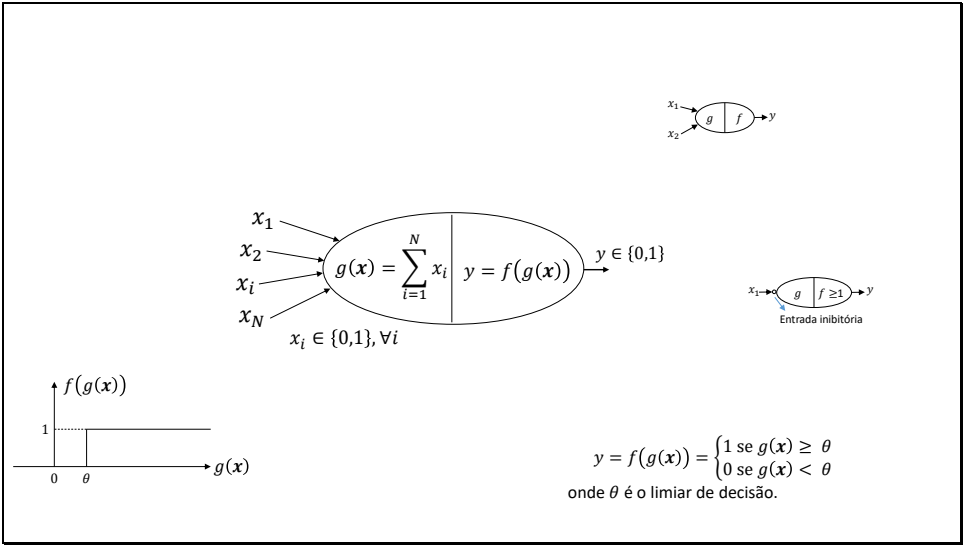
TO PROVE YOU'RE A HUMAN, CLICK ON ALL THE PHOTOS THAT SHOW PLACES YOU WOULD RUN FOR SHELTER DURING A ROBOT UPRISING.

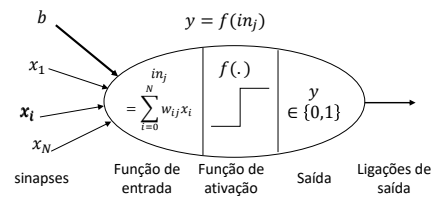


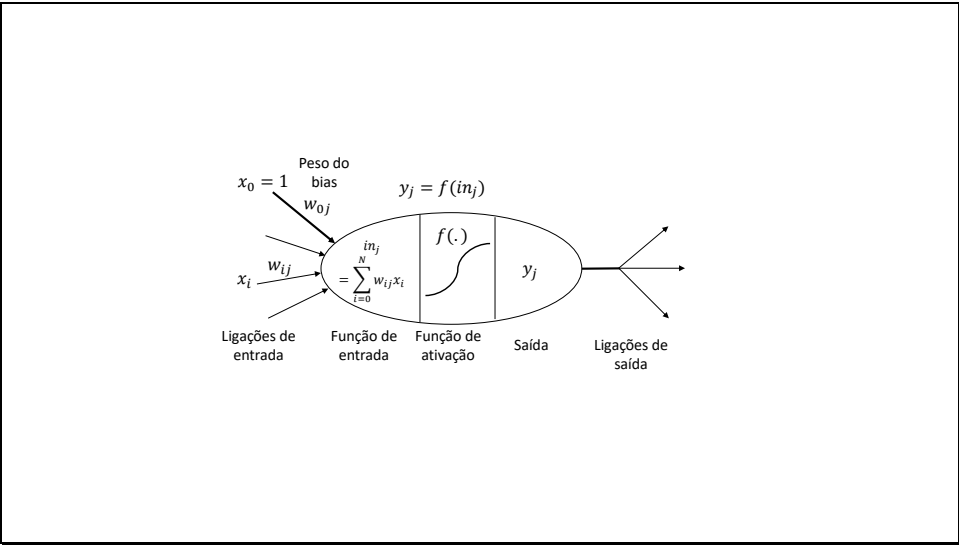
I DON'T USUALLY SPEND MY MONEY  
BUT WHEN I DO I DUMP ALL OF THEM ON GPUS

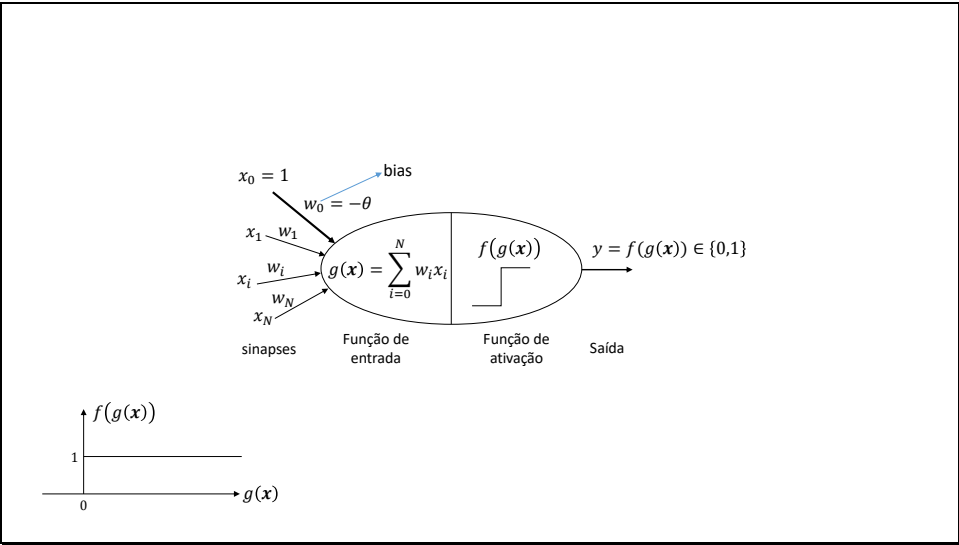
Figuras

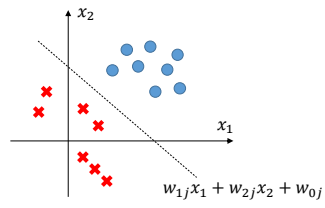












Slide 21

