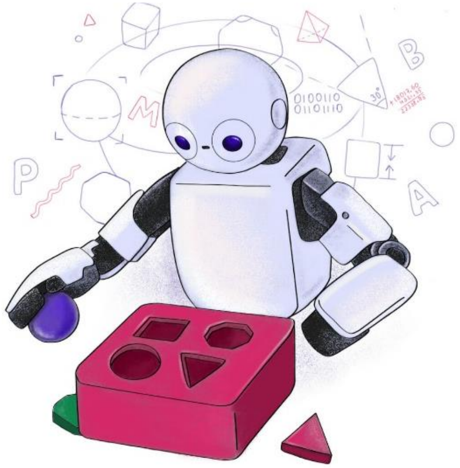
TP555 - Inteligência Artificial e Machine Learning:
Redes Neurais Artificiais (Parte I)



Inatel

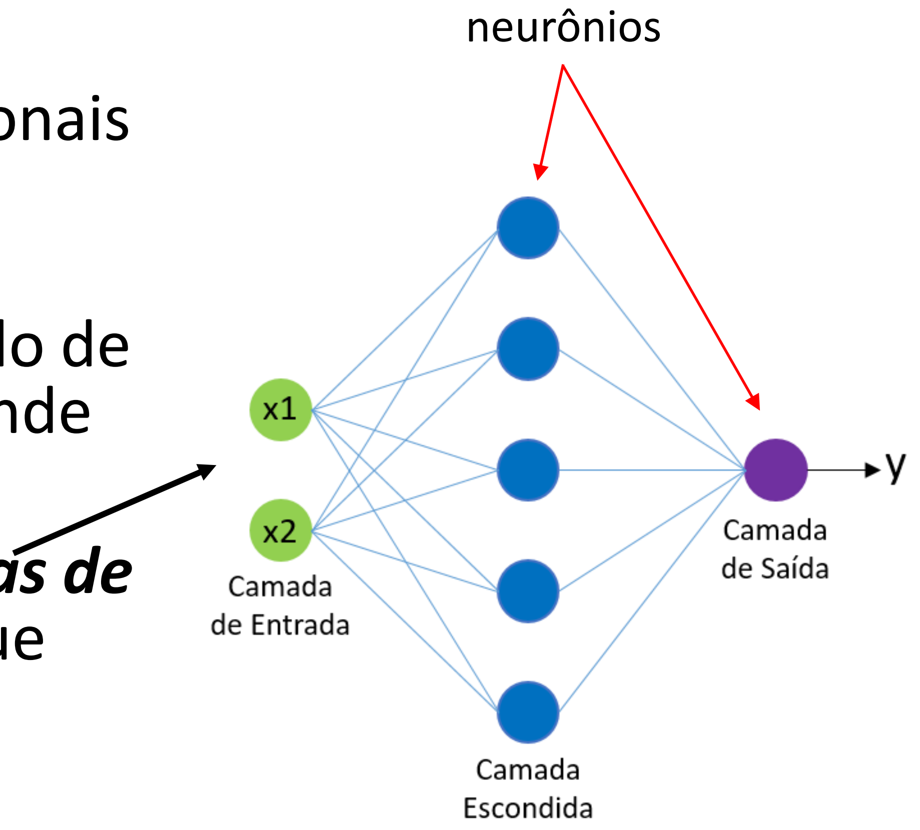
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Introdução

- Vamos falar sobre um tópico que parece, inicialmente, não ser relacionado com a disciplina: o cérebro.
- Entretanto, como veremos a seguir, as idéias que discutimos até agora são úteis na ***construção de modelos matemáticos que aproximam a atividade de aprendizado do cérebro***.
- E como veremos, essas ideias que já discutimos, nos ajudarão a entender o funcionamento das redes neurais artificiais (RNAs).
- Redes neurais artificiais são uma das formas mais populares e efetivas para implementação de sistemas de aprendizado de máquina e mereceriam por sí só uma disciplina em separado.
- Neste tópico veremos uma breve visão geral sobre as RNAs.

Redes Neurais Artificiais

- ***Redes neurais artificiais*** são modelos computacionais inspirados pelo funcionamento do cérebro dos animais.
- Elas são capazes de realizar tarefas de aprendizado de máquina (e.g., regressão e classificação) com grande eficácia.
- RNAs são geralmente apresentadas como ***sistemas de nós (unidades ou neurônios)*** ***interconectados***, que calculam valores de saída, simulando o comportamento de ***redes neurais biológicas***.
- Esta primeira parte deste tópico, foca nos elementos básicos de construção de uma rede neural, os ***nós*** ou ***neurônios***.



Algumas aplicações famosas

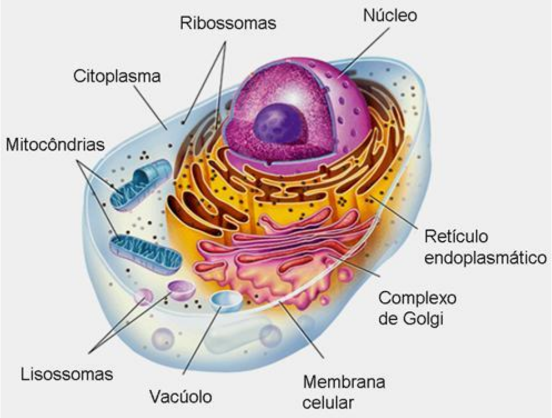
- RNAs são versáteis, poderosas e escalonáveis, tornando-as ideais para realizar tarefas grandes e altamente complexas de ***aprendizado de máquina***, como por exemplo:
  - Classificar bilhões de imagens (e.g., Google Images, Facebook, etc.),
  - Assistentes virtuais inteligentes (e.g., chatGPT da Open AI, Siri da Apple, Alexa da Amazon e Google Assistant),
  - Recomendar vídeos que melhor se adequam ao comportamento de centenas de milhões de usuários (e.g., YouTube, Netflix),
  - Pilotar um veículo com pouca ou nenhuma intervenção humana.
  - Reconhecimento facial para desbloquear celulares com o rosto (e.g., apple face ID).

Alexa

Google Assistant

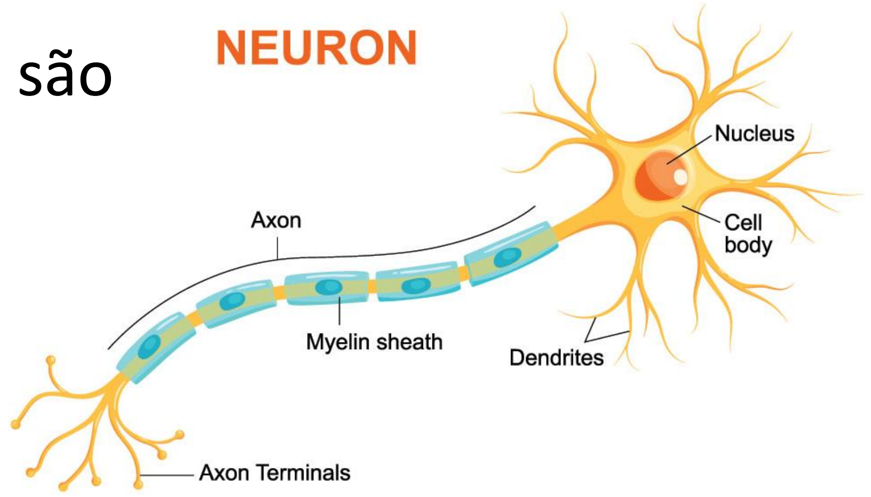
Siri

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***.
- Células podem ser classificadas em ***procariontes*** e ***eucariontes***.
- Células ***procariontes*** têm uma ***estrutura simples e não possuem núcleo*** (e.g., bactérias).
- As células ***eucariontes*** (plantas, animais, fungos, protozoários, algas, e amebas) possuem três partes principais: ***membrana, citoplasma*** e ***núcleo***.
  - A ***membrana*** "delimita a célula", i.e., ela isola seu interior do meio externo.
  - O ***citoplasma*** é o espaço intracelular entre a membrana e o núcleo.
    - Ele é preenchido pelo ***citosol*** onde estão suspensas as ***organelas*** (e.g., mitocôndrias, lisossomos, etc.).
  - Já o ***núcleo*** ***controla as atividades celulares e armazena a maior parte da informação genética (DNA) da célula***.

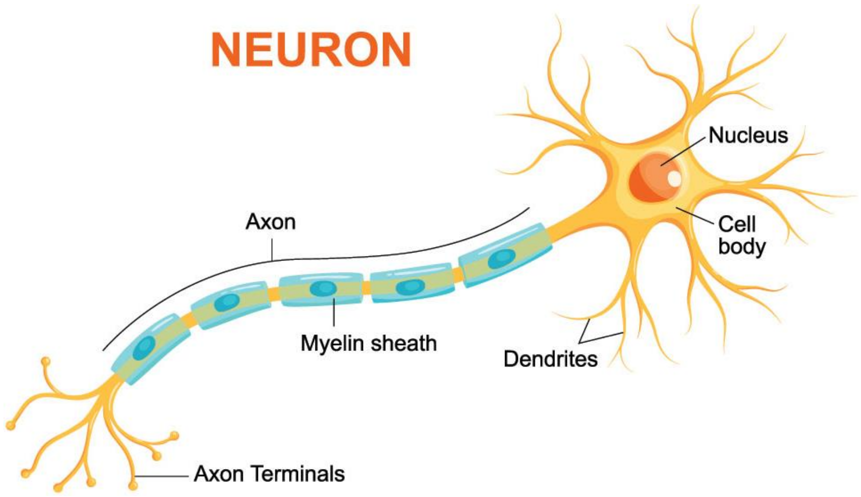
Um pouco de contexto



- Os ***neurônios*** são células ***eucariontes*** também, mas são células que ***possuem mecanismos eletroquímicos*** característicos.
- Os neurônios apresentam três partes básicas: os ***dendritos***, o ***axônio*** e o ***corpo celular (soma)***.
- Os ***dendritos*** são prolongamentos do neurônio que garantem a ***recepção de estímulos de outros neurônios***, levando impulsos nervosos em direção ao ***corpo celular***.
- O ***axônio*** é um prolongamento que garante o ***envio de informação (estímulos) a outros neurônios*** através de seus ***terminais***.
- Cada neurônio possui apenas um axônio, o qual é, geralmente, mais longo que os dendritos.

Um pouco de contexto

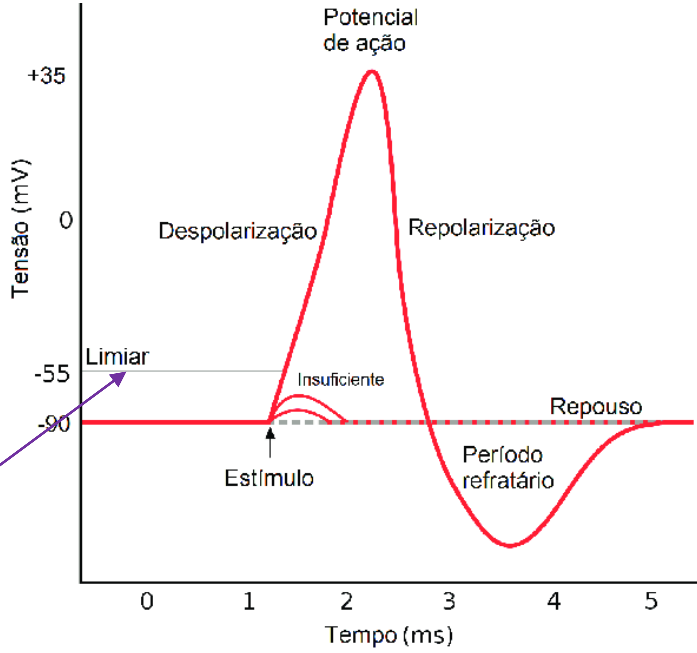
- O ***corpo celular*** (também conhecido como ***soma***) contém o núcleo do neurônio e é responsável por realizar a ***integração dos estímulos recebidos pelo neurônio através de seus dendritos***.
- Os pontos de contato entre os dentritos de um neurônio e os terminais do axônio de outro neurônio são chamados de ***sinapses***.
- Ou seja, os ***neurônios se comunicam uns com os outros através das sinapses***.
- Sinapses podem ser químicas, as mais comuns, ou elétricas, muito pouco comuns.
- As figuras ao lado mostram um ***neurônio*** e uma sinapse química.

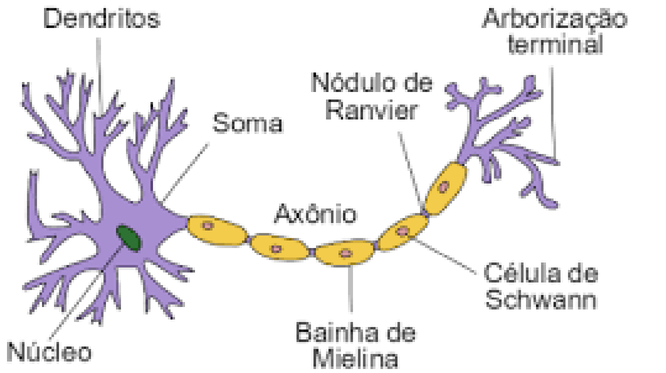


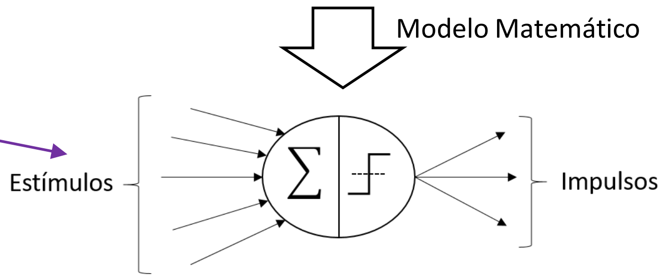
Sinapse química

Um pouco de contexto

- Em termos bem simples, mas lembrando de que existem exceções, nós podemos simplificar o funcionamento do ***neurônio*** como:
  - O neurônio recebe estímulos elétricos, basicamente a partir dos dendritos.
  - Esses estímulos são somados no corpo celular (*soma*).
  - Se a soma dos estímulos exceder um certo ***limiar de ativação***, o ***neurônio*** gera um pulso (ou ***potencial de ação***) que é enviado pelos terminais do axônio a outros neurônios.
- Um ***neurônio*** pode se conectar a até 20.000 outros ***neurônios*** através das ***sinapses***.
- Sinais são passados de ***neurônio*** para ***neurônio*** através de ***reações eletroquímicas***.
- Do ponto de vista do nosso curso, o ***neurônio*** será considerado como um ***sistema com várias entradas e uma ou mais saídas*** onde a comunicação entre neurônios é feita através de sinais elétricos.



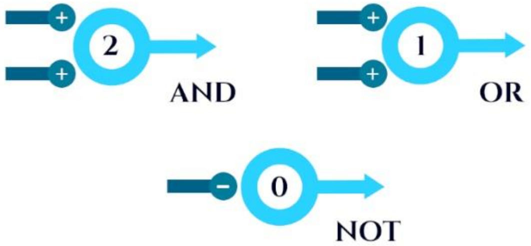




O Modelo de McCulloch e Pitts

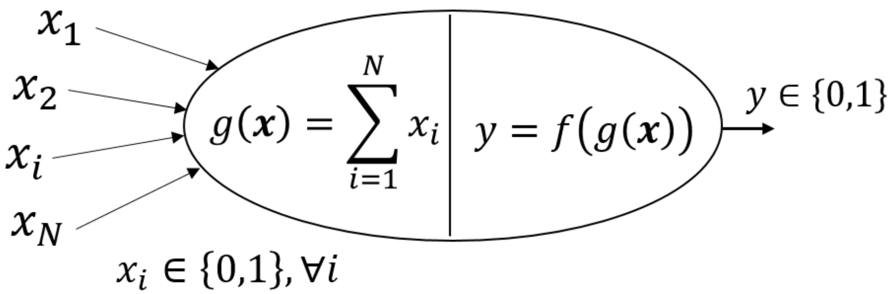
- Em 1943, a partir do entendimento do funcionamento dos neurônios, Warren McCulloch e Walter Pitts apresentam em um artigo o primeiro ***modelo computacional de um neurônio***.
- A partir desse modelo, foi possível estabelecer uma conexão entre o funcionamento de um neurônio e a ***lógica proposicional***.
- ***Lógica proposicional*** se baseia em ***proposições***.
  - Uma ***proposição*** é uma ***sentença declarativa*** ou ***afirmação***, ou seja, é uma sentença que faz uma ***afirmação*** sobre um fato, podendo este ser verdadeiro ou falso.
- Existe uma correspondência direta entre a lógica proposicional e a lógica Booleana.
  - Podemos pensar em uma ***sentença declarativa*** como sendo uma ***expressão Booleana***
    - 1 ou 1 = 1
    - 1 e 0 = 0
- O artigo de McCulloch e Pitts fornece *insights* fundamentais sobre como a ***lógica proposicional*** pode ser processada por um neurônio.
- A partir daí, a relação com a computação foi direta e natural.

Walter Pitts e Warren McCulloch



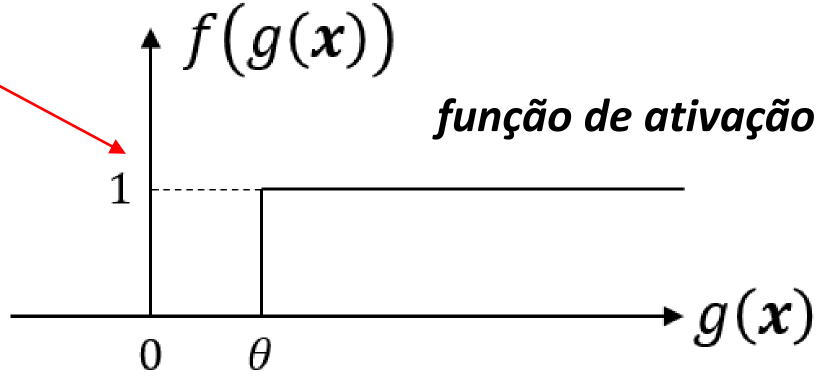
O Modelo de McCulloch e Pitts

- A figura ao lado apresenta o modelo matemático do ***neurônio*** proposto por McCulloch e Pitts.
- Grosso modo, o ***neurônio*** é ativado (ou disparado) quando a ***soma*** de suas entradas excede o ***limiar de ativação***, θ.
- As suposições do modelo de McCulloch e Pitts (M-P) são:
  - Os valores das entradas, $x_i, \forall i$, ou também chamados de ***sinapses***, são sempre valores booleanos, i.e., '0', ou '1'.
  - As entradas são multiplicadas por pesos unitários (+/- 1) e somadas.
  - A atividade do ***neurônio*** é um processo do tipo "***tudo ou nada***", ou seja, um processo binário (0 ou 1).
  - Portanto, a ***função de ativação*** do neurônio é uma ***função degrau*** com ***ponto de disparo*** dependente do ***limiar de ativação***, θ.
  - Um certo número de ***sinapses*** deve ser excitado para que o neurônio "dispare".
- O modelo do ***neurônio*** de McCulloch e Pitts nada mais é do que um ***classificador linear com limiar de decisão rígido***, ***pesos unitários*** e ***atributos booleanos***.



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

onde θ é o ***limiar de ativação***.

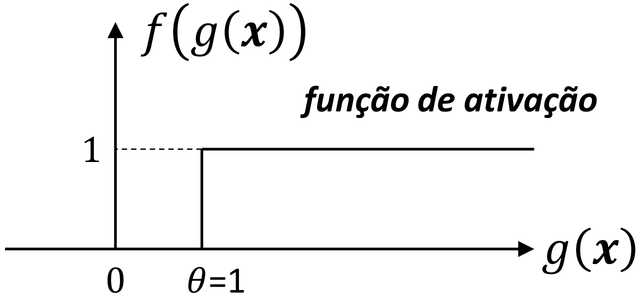


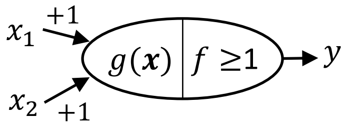
Exemplos de portas lógicas com o modelo M-P

Podem ser interpretados como problemas de classificação binária.

OR			
x_1	x_2	$g(\boldsymbol{x})$	y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

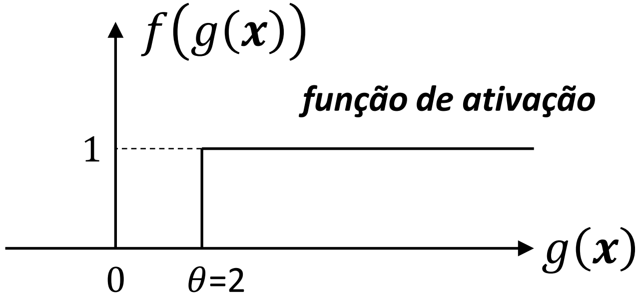
- Qual é o valor do ***limiar de ativação***, θ?
- Analisando-se $g(\boldsymbol{x})$, vemos que o disparo deve ocorrer quando $g(\boldsymbol{x}) \geq 1$, portanto, $\theta = 1$.

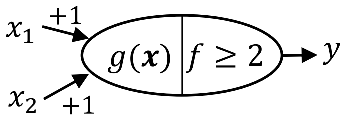




AND			
x_1	x_2	$g(\boldsymbol{x})$	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

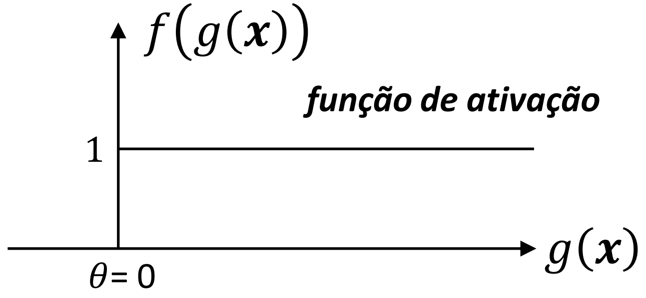
- Qual é o valor do ***limiar de ativação***, θ?
- Analisando-se $g(\boldsymbol{x})$, vemos que o disparo deve ocorrer quando $g(\boldsymbol{x}) \geq 2$, portanto, $\theta = 2$.

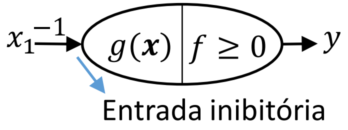




NOT			
x_1	$-x_1$	$g(\boldsymbol{x})$	y
0	0	0	1
1	-1	-1	0

- Qual é o valor do ***limiar de ativação***, θ?
- Analisando-se x_1, vemos que para o disparo ocorrer, seu valor deve ser ***negado*** (i.e., multiplicado por -1), e assim, o disparo ocorre quando $g(\boldsymbol{x}) \geq 0$, portanto, $\theta = 0$.





OBS.: Entradas inibitórias são entradas que tem seus valores multiplicados por -1.

Exemplos de portas lógicas com o modelo M-P

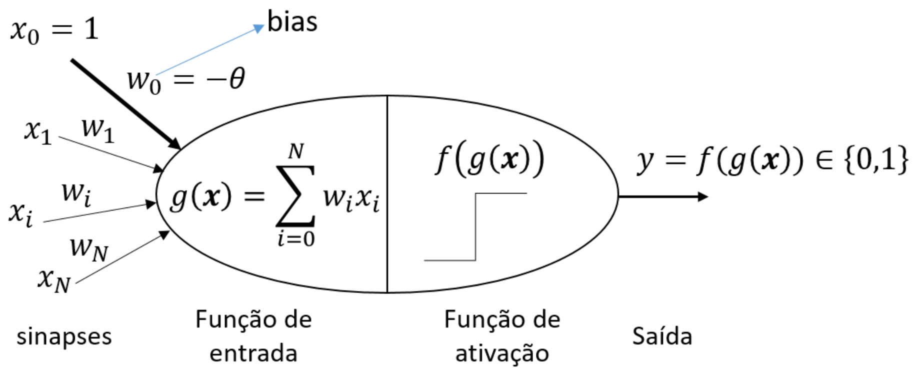
- Para casa:
  - Qual deve ser o valor do ***limiar de ativação***, θ, para a porta lógica XOR?

XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron

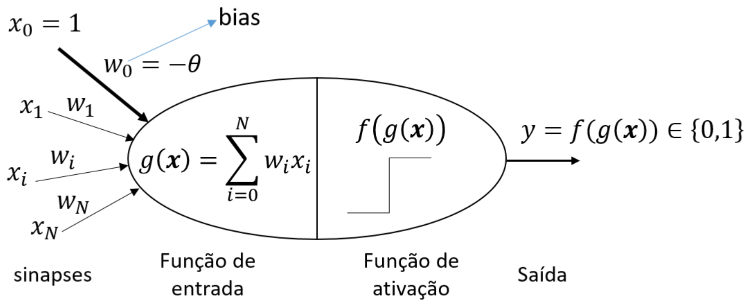
- Em 1958, Frank Rosenblatt, propôs um novo ***modelo computacional*** ***mais geral*** que o modelo do ***neurônio*** de McCulloch e Pitts.
- O modelo criado por ele é chamado de ***perceptron*** e é mostrado na figura ao lado.
- O ***perceptron*** ***introduz o conceito de aprendizado supervisionado***, onde os pesos do neurônio são ajustados iterativamente com base em uma ***regra de aprendizado***.
- Portanto, o ***perceptron*** é um modelo de ***aprendizado supervisionado*** para ***classificação binária***, ou seja ***problemas com duas classes***.
- Por definição, o ***perceptron*** só é capaz de classificar padrões ***linearmente separáveis***, assim como o modelo de M-P.

Frank Rosenblatt e o Mark I Perceptron, que foi treinado para reconhecer diferentes formas geométricas, como círculos, quadrados e triângulos.

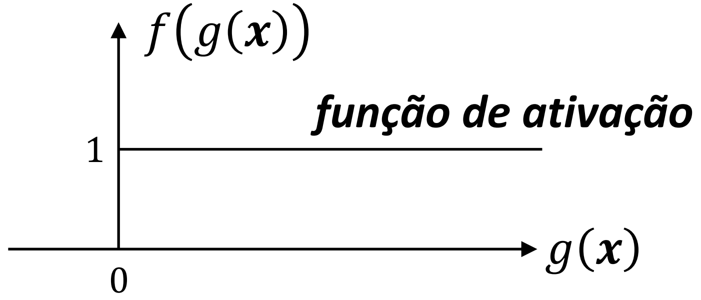


Perceptron

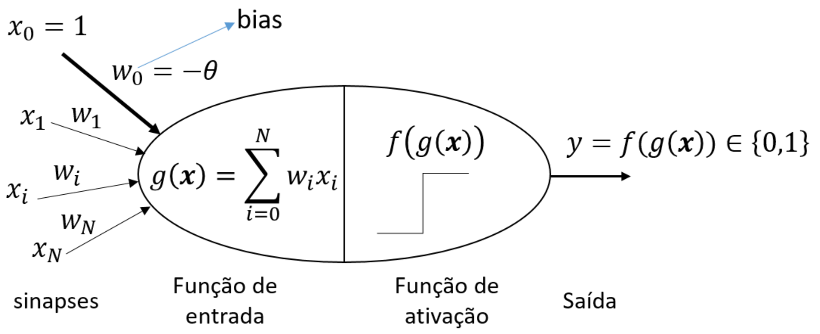
- Esse novo modelo supera algumas das limitações do modelo de M-P:
  - Introdução do conceito de ***pesos sinápticos*** (uma medida de importância dos atributos) para as entradas (ou ***sinapses***).
  - Entradas com valores reais, não sendo mais limitadas a valores booleanos como no modelo de M-P.
  - E um método para que o modelo aprenda os ***pesos***.
- Essas novas características tornam esse modelo mais útil e generalizado.
- Entretanto, assim como no modelo de M-P, a ***função de ativação*** utilizada pelo ***perceptron*** também é a ***função degrau***, com a diferença que o ***ponto de disparo*** não varia com o ***limiar de ativação***, θ, ele é sempre fixo em 0.
- Veremos na sequência o motivo disso.

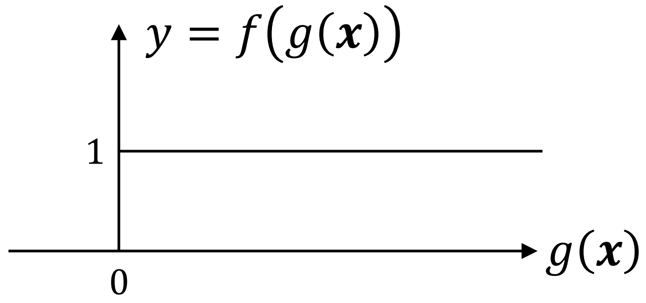


$$y = f(g(\boldsymbol{x})) = \begin{cases} 1, \text{se } g(\boldsymbol{x}) \geq 0 \\ 0, \text{se } g(\boldsymbol{x}) < 0 \end{cases}$$



Perceptron





- Note que o ***limiar de ativação***, θ, agora é um dos pesos, chamado de ***peso de bias***, w_0.
  - Isso é feito para que θ seja ***aprendido*** junto com os outros pesos.
- Lembre-se do modelo de M-P que a ativação, i.e., $y = 1$, ocorre quando $\sum_{i=1}^{N} w_i x_i \geq \theta$, mas se trouxermos θ para o lado esquerdo da expressão, passamos a ter

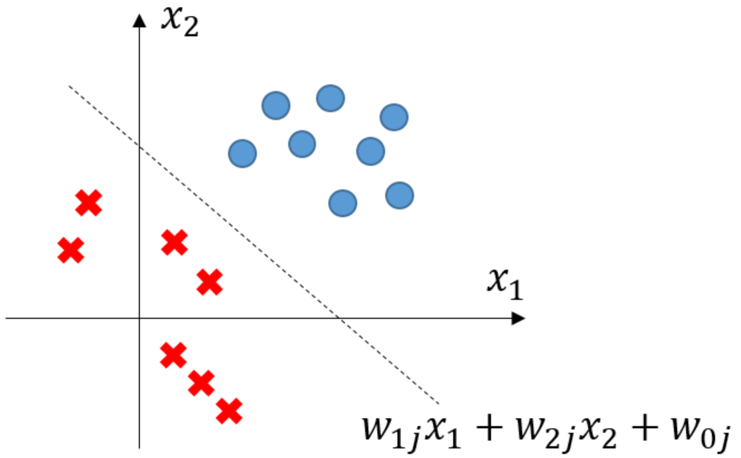
$$\sum_{i=1}^{N} w_i x_i - \theta \geq 0.$$

- Se criarmos um atributo $x_0 = 1$ e fizermos $w_0 = -\theta$, temos que a combinação linear $g(\boldsymbol{x}) = \sum_{i=0}^{N} w_i x_i$ deve ser maior ou igual a 0 para que haja a ativação.
- Isso pode ser expresso por meio de uma ***função de ativação*** do tipo ***degrau com transição fixa em zero***.
- Assim, o ***limiar de ativação*** passa a ser controlado pelo valor do ***peso do bias***, w_0.
  - O limiar de ativação é ajustado indiretamente através da atualização do peso de bias, w_0.
- O tipo de resposta do ***perceptron*** dá origem a um ***classificador binário***, ou seja, para ***problemas com duas classes***.
- As classes são separadas por uma ***fronteira de decisão linear*** para o qual a equação (***função discriminante***) abaixo é verdadeira.

$$g(\boldsymbol{x}) = \sum_{i=0}^{N} w_i x_i = 0.$$

Perceptron

- No ***espaço de atributos*** definido por $x_i, \forall i$, $g(\boldsymbol{x})$ é a equação de um ***hiperplano*** (ponto, reta, plano, etc., dependendo do número de dimensões).
- Portanto, um ***perceptron***, por definição, só é capaz de ***classificar perfeitamente*** dados que sejam ***linearmente separáveis*** (ou seja, separáveis por um ***hiperplano***).
- O ***perceptron*** convergirá apenas se o conjunto de dados for ***linearmente separável***.
  - Classes suficientemente espaçadas de tal forma que um hiperplano as separe.
- 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 da 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***.



Regra de aprendizado do perceptron

- Como discutimos anteriormente, a ***função degrau*** tem derivada igual a 0 em todos os pontos, exceto em 0, onde ela é indefinida.
- Portanto, nós não podemos utilizar o ***gradiente descentende*** para treinar o ***perceptron***.
- Existe, porém, uma ***regra simples e intuitiva 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 em um 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

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

Regra de aprendizado do perceptron

- Em seguida, calcula-se o erro entre a saída $\hat{y}$ do ***perceptron*** e o rótulo y (valor esperado) do exemplo:

$$e = y - \hat{y}.$$

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

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

onde α 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 (i.e., uma época).
- 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 aplicada a um exemplo de entrada por vez. Os exemplos são escolhidos aleatóriamente, assim como feito com o ***gradiente descendente estocástico***.

Regra de aprendizado do perceptron

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(y - \hat{y})\boldsymbol{x}$$

- A ***equação de atualização dos pesos sinápticos*** é idêntica à equação de atualização dos pesos com o gradiente descendente estocástico, mas funciona de forma diferente, como vemos a seguir.
- Como ambos, o rótulo, y, e o valor de saída do perceptron, $\hat{y}$, assumem apenas 2 valores, 0 ou 1, existem apenas 3 possibilidades para a equação de atualização dos pesos:
    1. Se a saída for correta, i.e., $y = \hat{y}$, então os pesos não são atualizados.
    2. Se y = 1, mas $\hat{y}$ = 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 $\boldsymbol{w}^T\boldsymbol{x}$ aumente tal que y se torne 1.
    3. Se y = 0, mas $\hat{y}$= 1, então o valor do peso é diminuido 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 $\boldsymbol{w}^T\boldsymbol{x}$ diminua tal que y se torne 0.

Exemplo: Perceptron com SciKit-Learn

```python
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(fit_intercept=False, 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 a partir dos dados originais. Função lógica AND.

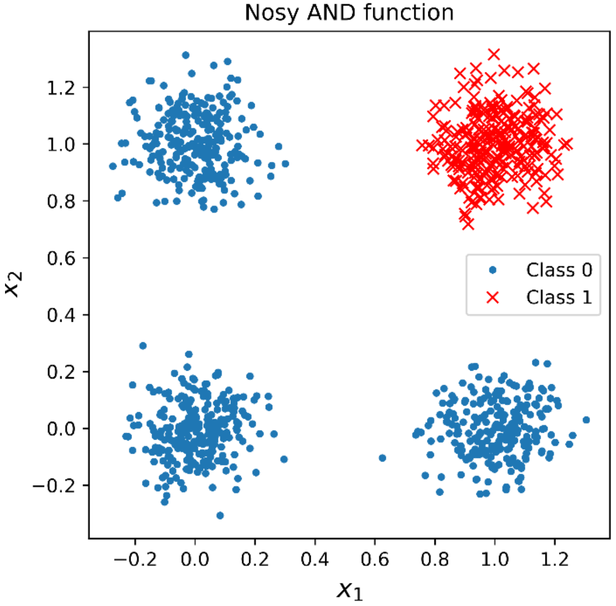
Adiciona ruído aos atributos de entrada

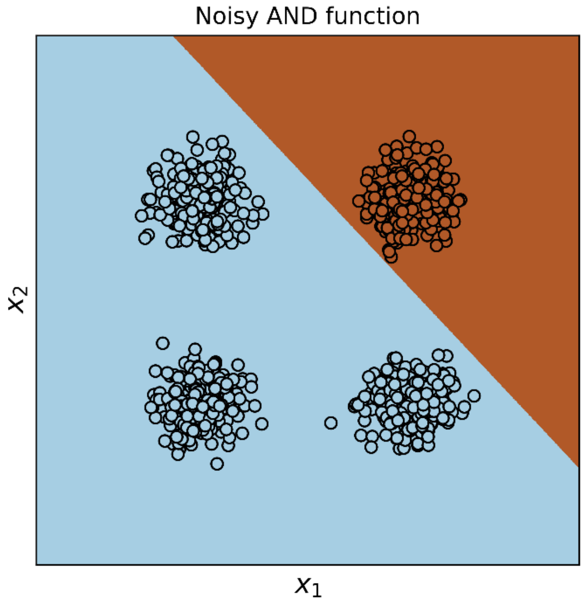
Cria vetor de 1s para o peso de bias.

Instancia e treina o Perceptron.

Realiza a predição.

Calcula erro quadrático médio.





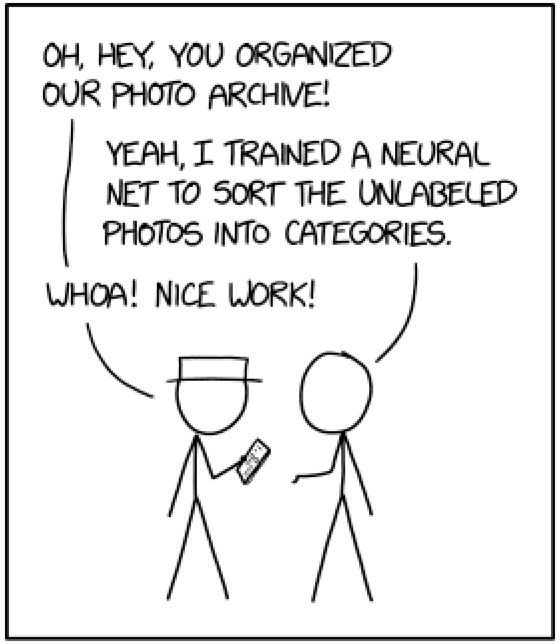
- Exemplo de classificação de dados ruidosos linearmente separáveis.
- A base de dados é gerada a partir da função de uma porta lógica AND com ruído Gaussiano adicionado às amostras.
- Como podemos ver, o perceptron classifica perfeitamente o conjunto de dados ruidosos.

Exemplo: Perceptron.ipynb

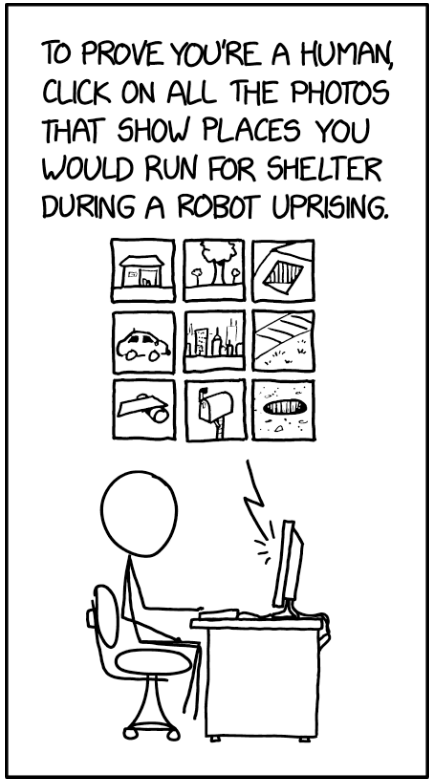
Avisos

- Vocês já podem fazer os exercícios da lista #10.

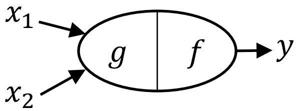
Obrigado!

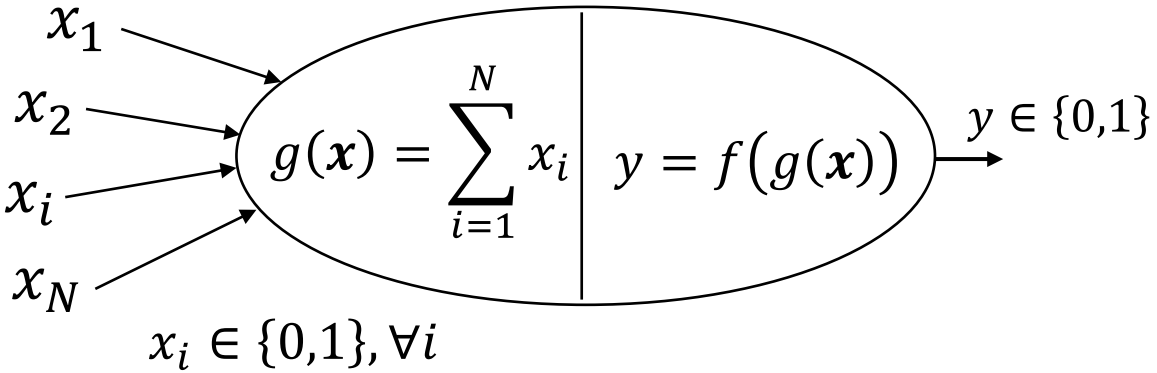


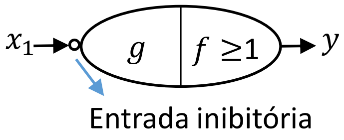
ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

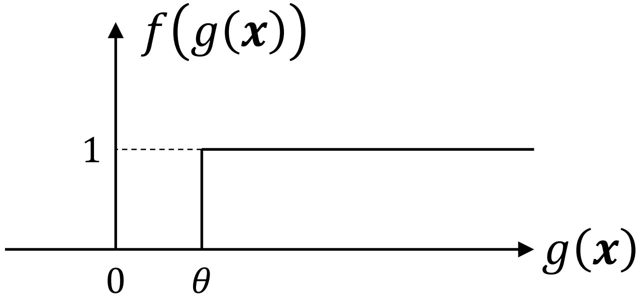


Figuras



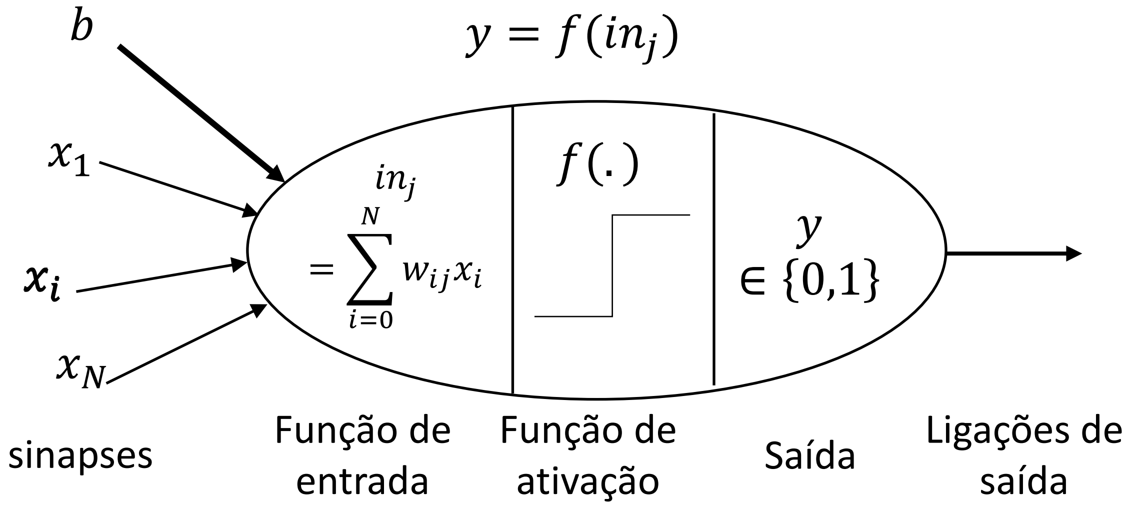




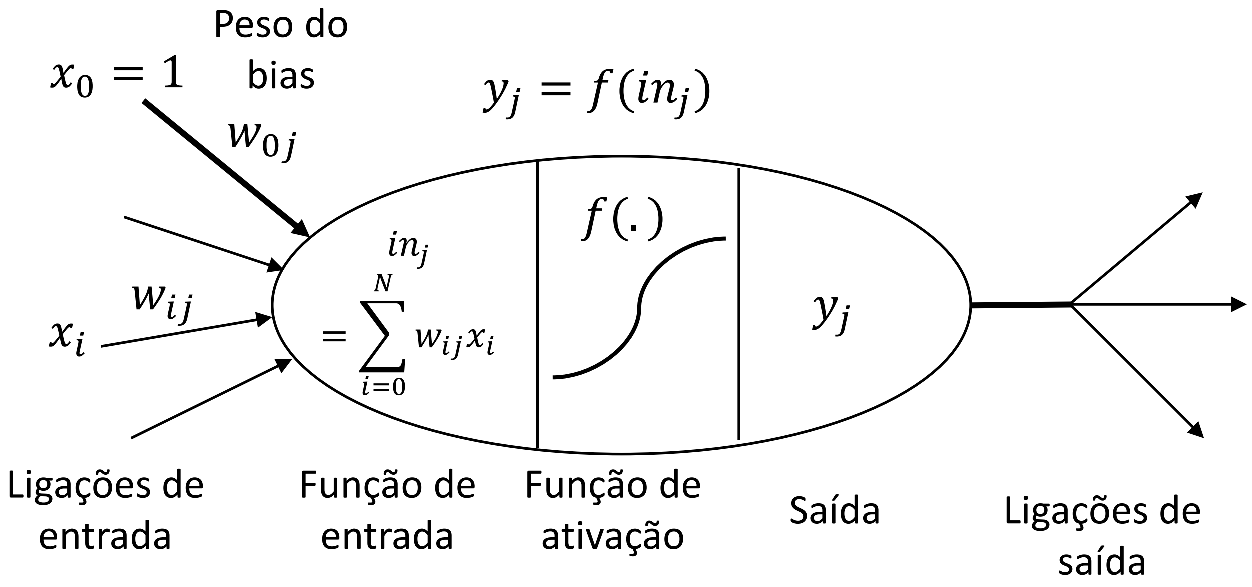


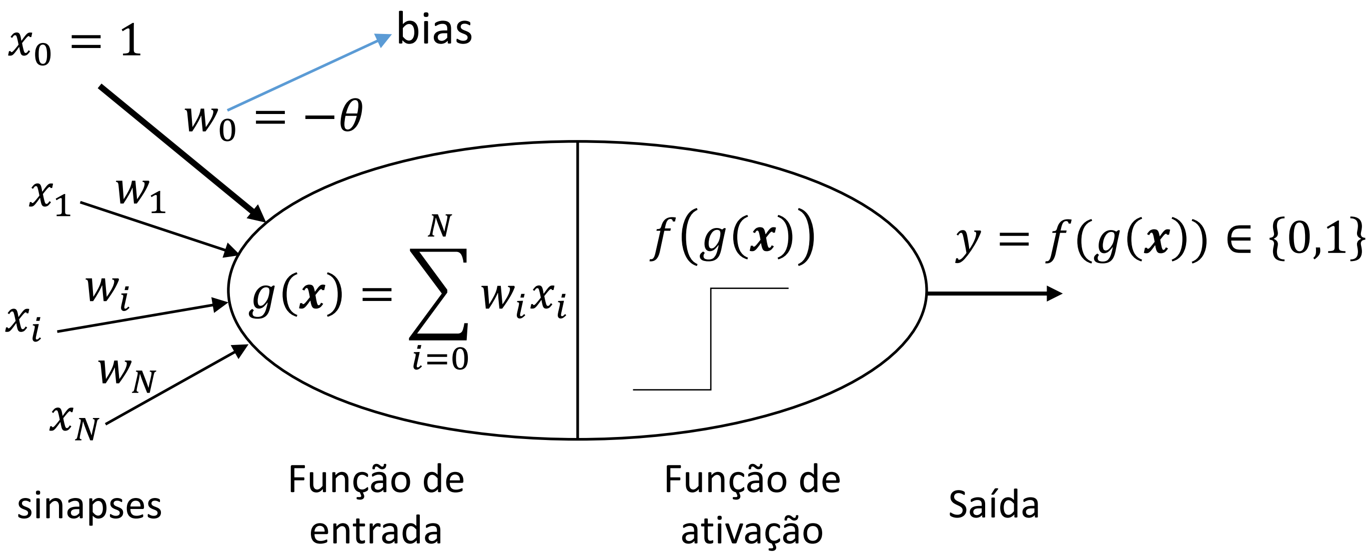
$$y = f\big(g(\boldsymbol{x})\big) = \begin{cases} 1 \text{ se } g(\boldsymbol{x}) \geq \theta \\ 0 \text{ se } g(\boldsymbol{x}) < \theta \end{cases}$$

onde θ é o limiar de decisão.

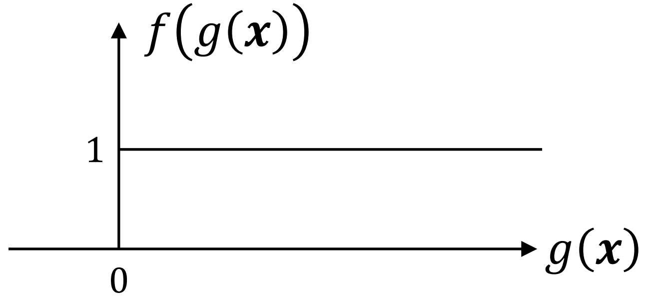


b
$y = f(in_j)$
x_1
x_i
x_N
$in_j = \sum_{i=0}^{N} w_{ij} x_i$
$f(.)$
$y \in \{0,1\}$
sinapses
Função de entrada
Função de ativação
Saída
Ligações de saída

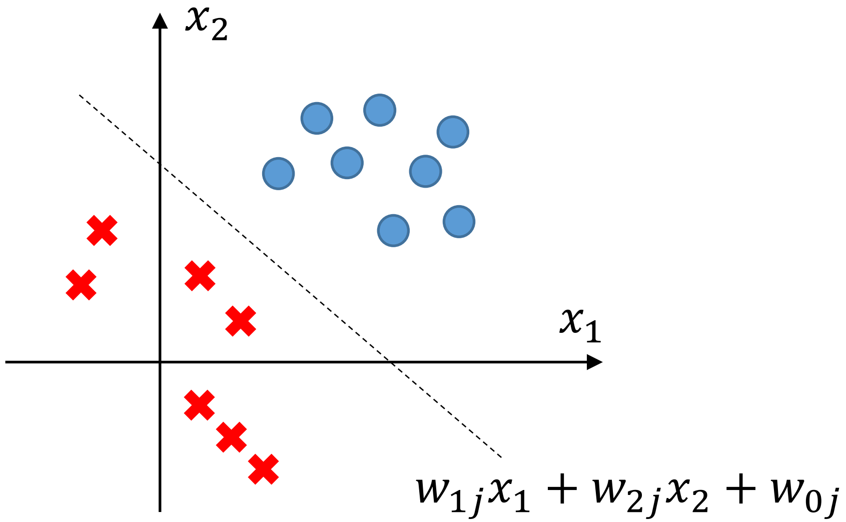

Peso do bias
$x_0 = 1$
w_{0j}
$y_j = f(in_j)$
$in_j = \sum_{i=0}^{N} w_{ij} x_i$
$f(.)$
y_j
w_{ij}
x_i
Ligações de entrada
Função de entrada
Função de ativação
Saída
Ligações de saída



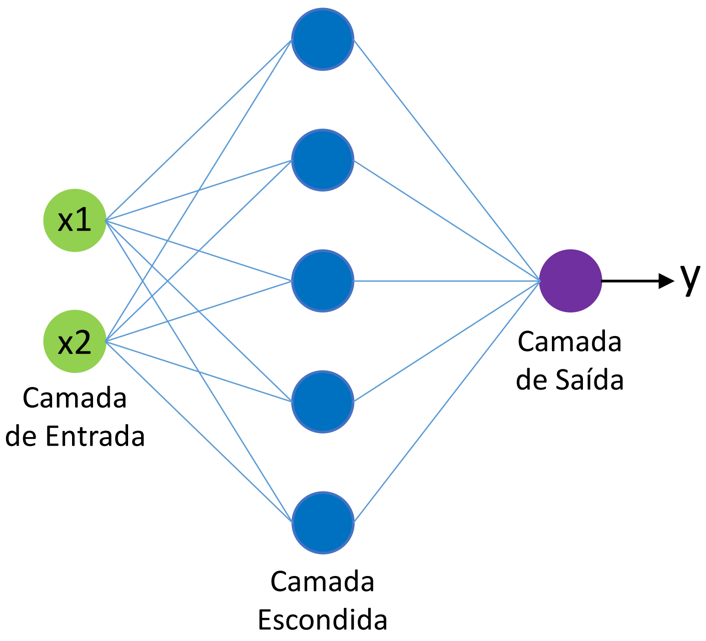
$x_0 = 1$
bias
$w_0 = -\theta$
x_1 w_1
w_i
x_i
w_N
x_N
$g(\boldsymbol{x}) = \sum_{i=0}^{N} w_i x_i$
$f\big(g(\boldsymbol{x})\big)$
$y = f(g(\boldsymbol{x})) \in \{0,1\}$
sinapses
Função de entrada
Função de ativação
Saída



$f\big(g(\boldsymbol{x})\big)$
1
0
$g(\boldsymbol{x})$



x_2
x_1
$w_{1j}x_1 + w_{2j}x_2 + w_{0j}$


x1
x2
Camada
de Entrada
Camada
Escondida
Camada
de Saída
y

XNOR