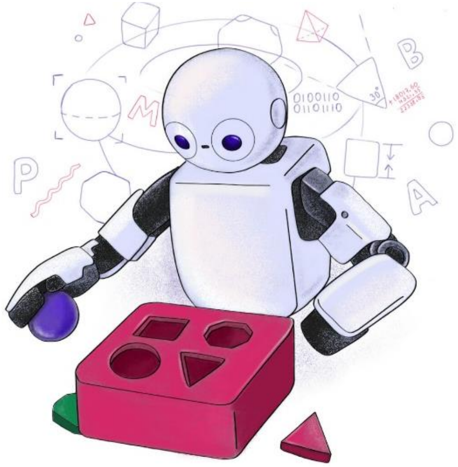
TP555 - Inteligência Artificial e Machine Learning:

Redes Neurais Artificiais (Parte I)



Inatel

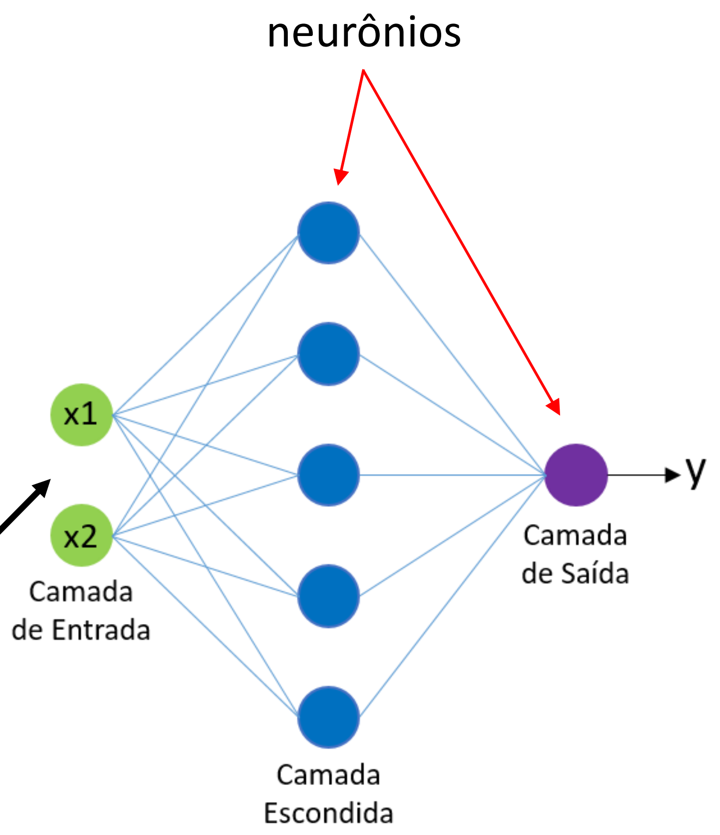
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

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 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) interconectados***, que computam 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 ***neurônios***.

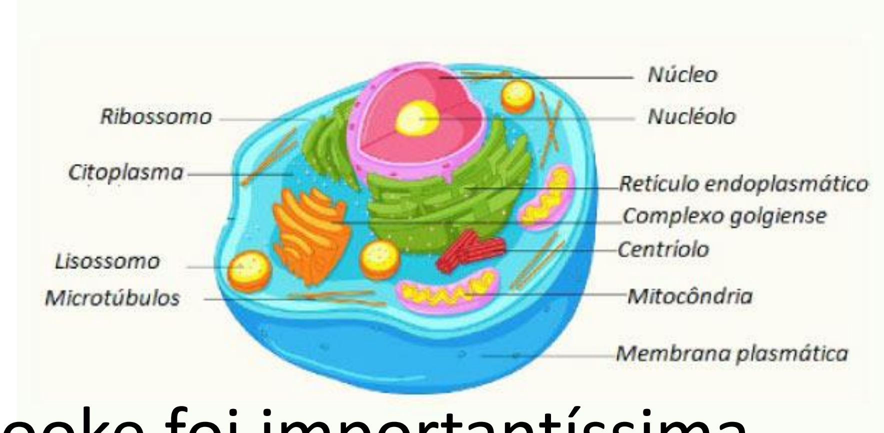


Algumas aplicações famosas

- 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, como o Google Images faz),
  - serviços de reconhecimento de fala (por exemplo, o Siri da Apple, Alexa da Amazon e Google Assistant da Google),
  - recomendar vídeos que melhor se adequam ao comportamento de centenas de milhões de usuários todos os dias (por exemplo, YouTube, Netflix),
  - ou aprender a vencer o campeão mundial de Go examinando milhões de partidas anteriores e depois jogando contra si mesmo (AlphaGo da DeepMind).

Alexa | Google Assistant | Siri

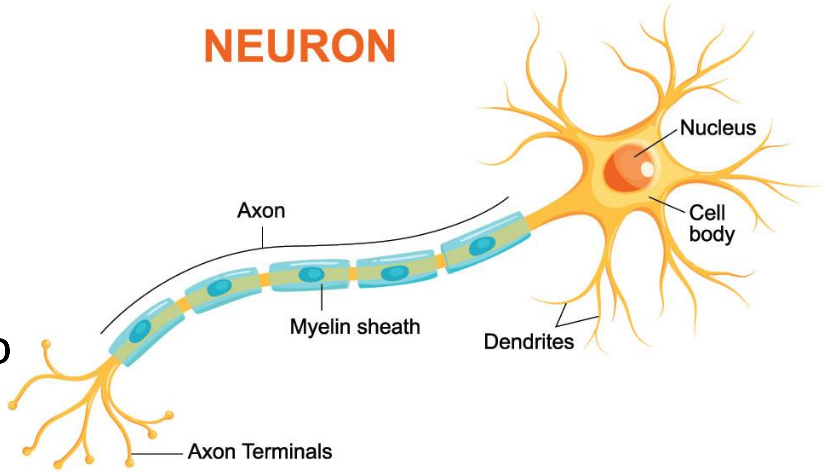
Um pouco de contexto



- 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***".
- As células ***eucariontes*** (plantas, animais, fungos, protozoários, e algas) 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***.
- Já o ***núcleo*** abriga a maior parte do material genético (DNA) da célula. Ele regula o metabolismo e armazena as informações genéticas da célula.

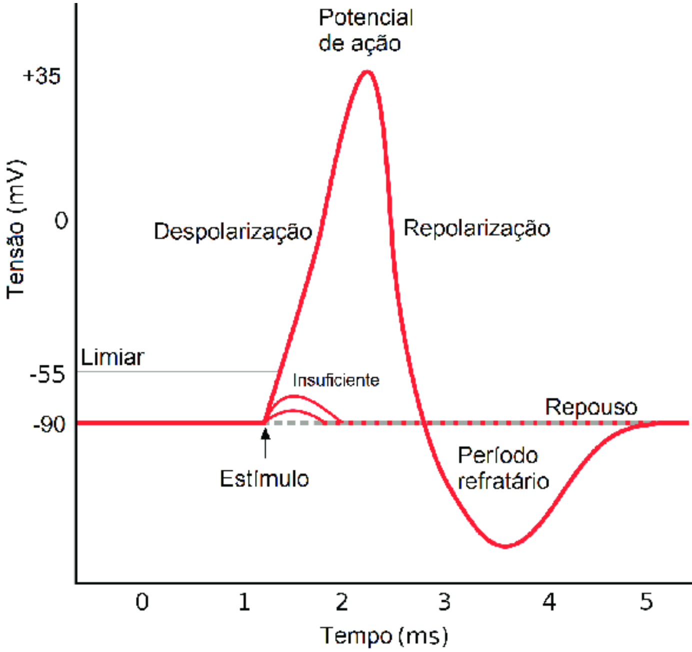
Um pouco de contexto

- Os ***neurônios*** são células ***eucariontes*** também, mas são células que possuem mecanismos elétricos e/ou químicos característicos.
- Os neurônios apresentam três partes básicas: os ***dendritos***, o ***axônio*** e o ***corpo celular***.
- 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.
- 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 locais/pontos de contato entre os dentritos de um neurônio e os terminais do axônio de outro neurônio são chamados de ***sinapses*** e os contatos entre eles de ***contatos sinápticos***.
- Ou seja, os neurônios se comunicam uns com os outros através das ***sinapses***.
- 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 no corpo celular (*soma*).
    - A integração dos estímulos pode levar à geração ou não de uma resposta elétrica enviada pelo axônio a outros neurônios.
- 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 100.000 outros ***neurônios*** através das ***sinapses***.
- Sinais são passados de ***neurônio*** para ***neurônio*** através de reações eletro-químicas.
- Do ponto de vista do nosso curso, o ***neurônio*** será considerado como um sistema com várias entradas e uma saída onde a comunicação entre neurônios é feita através de sinais elétricos.



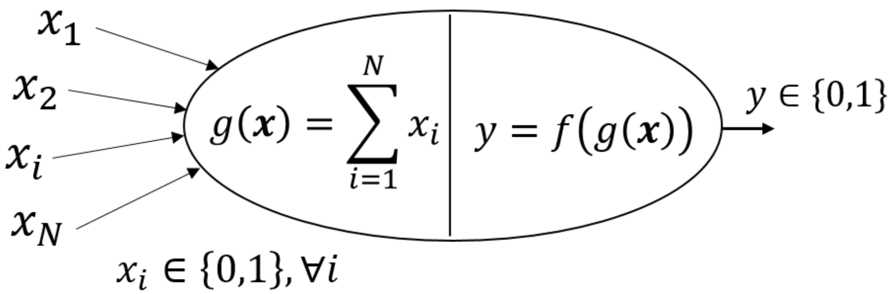
O Modelo de McCulloch e Pitts

- O final do século XIX e o início do século XX foram períodos fundamentais para o estabelecimento do conhecimento atual do sistema nervoso.
- De posse desse entendimento, em 1943, Warren McCulloch e Walter Pitts apresentaram 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 onde uma proposição é uma sentença declarativa, ou seja, é uma sentença que declara um fato podendo este ser verdeiro ou falso.
  - 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 natural.


Walter Pitts e Warren McCulloch

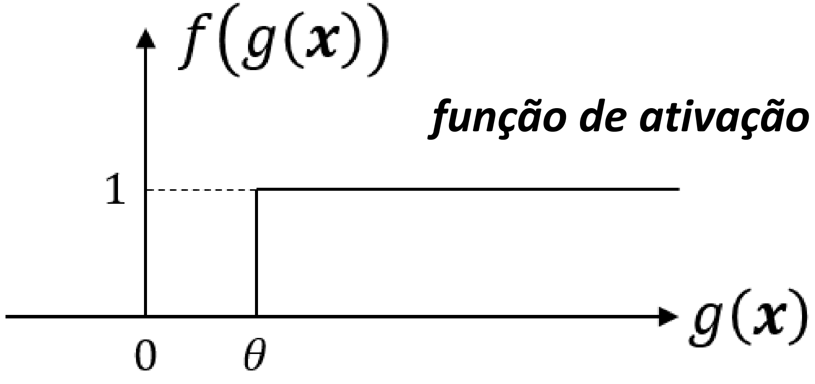
O Modelo de McCulloch e Pitts

- A figura ao lado mostra o modelo matemático do ***neurônio*** criado por McCulloch e Pitts.
- A grosso modo, o ***neurônio*** é ativado (ou disparado) quando uma ***combinação linear*** de suas entradas excede um ***limiar de ativação***.
- Ou seja, o modelo do ***neurônio*** de McCulloch e Pitts nada mais é do que um ***classificador linear com limiar de decisão rígido e pesos unitários***.
- 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***, θ**.**
  - Um certo número de ***sinapses*** deve ser excitado num determinado período para que o neurônio “dispare”.



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

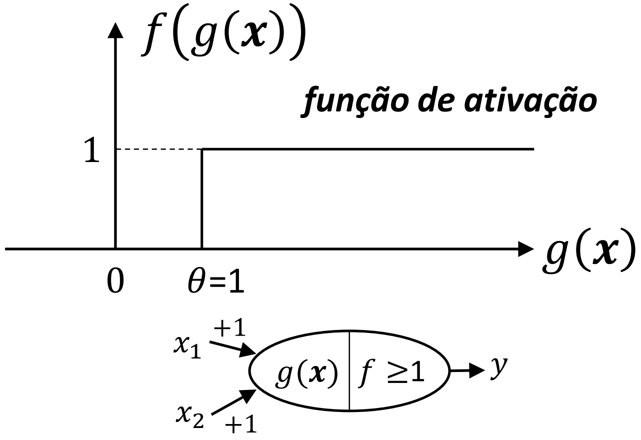
onde θ é o ***limiar de ativação***.



Exemplos com o modelo de McCulloch e Pitts

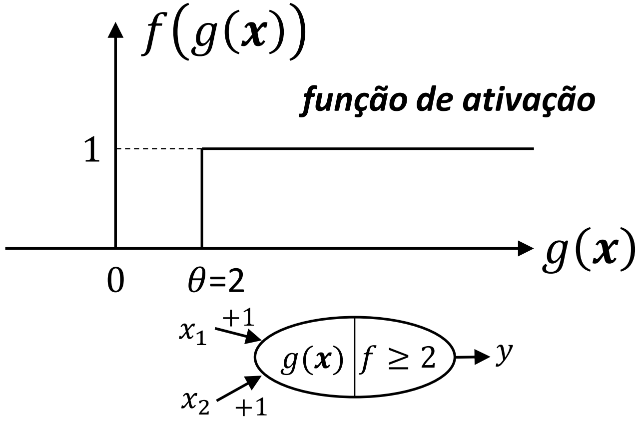
OR			
x_1	x_2	y	$g(\boldsymbol{x})$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	2

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



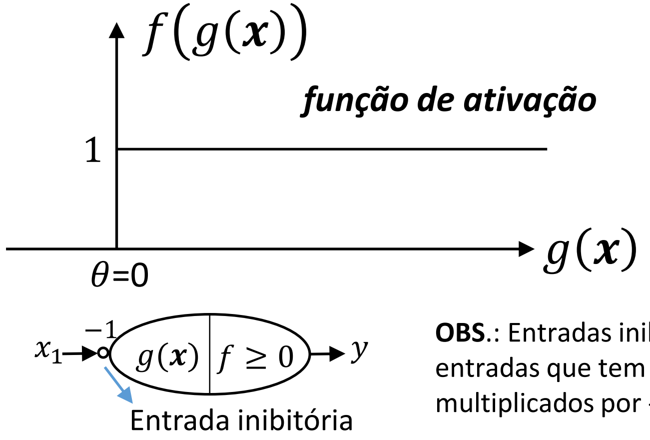
AND			
x_1	x_2	y	$g(\boldsymbol{x})$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	2

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



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

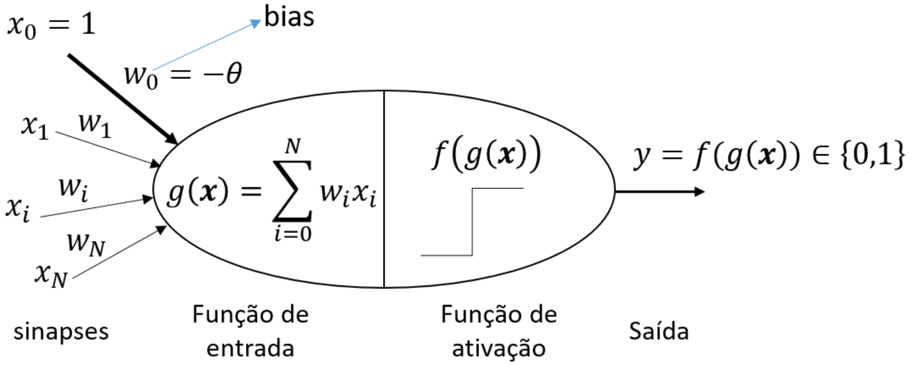
- Qual seria o valor do ***limiar de ativação***, θ?
- Analisando-se x_1, vemos que para o disparo ocorrer, seu valor deve ser ***inibido***, e assim, o disparo ocorre quando $g(\boldsymbol{x}) \geq 0$, portanto, $\theta = 0$.



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

Perceptron

- Em 1958, Frank Rosenblatt, propôs o modelo clássico do ***perceptron***.
- Em 1969, o modelo de Rosenblatt foi cuidadosamente analisado e refinado por Minsky e Papert.
- O modelo criado por eles é chamado de ***perceptron*** e é mostrado na figura ao lado.
- O modelo **perceptron**, é um modelo computacional mais geral que o modelo do ***neurônio*** de M-P.



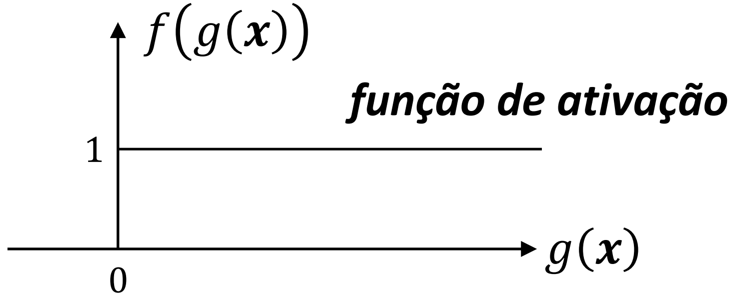
Perceptron

- 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***).
  - E um método para que o modelo aprenda os ***pesos***.
- Além disso, as entradas não são mais limitadas a valores booleanos, como no caso do modelo de M-P, suportando entradas com valores 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*** θ.

- 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***).
  - E um método para que o modelo aprenda os ***pesos***.
- Além disso, as entradas não são mais limitadas a valores booleanos, como no caso do modelo de M-P, suportando entradas com valores 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*** θ.

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

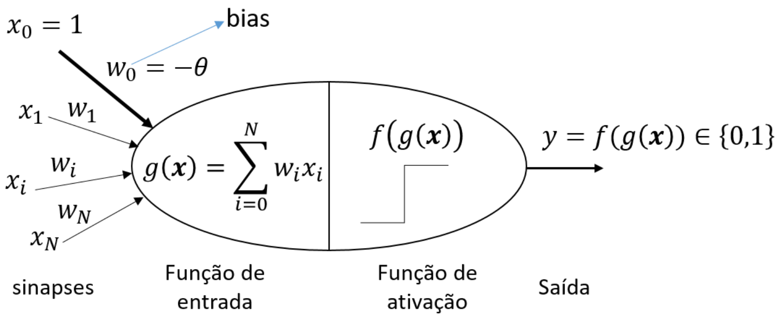
Perceba que o ***limiar de ativação*** θ agora faz parte das entradas e é chamado de ***bias***.

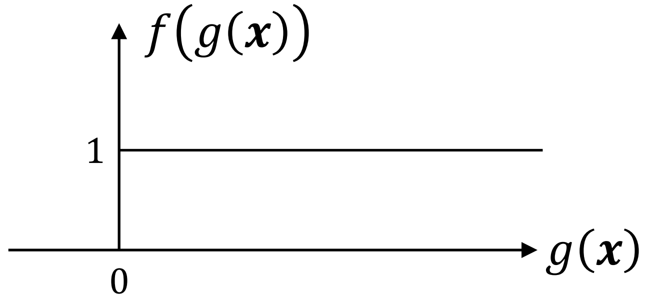


Perceptron

- 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*** $f(.)$ 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 limiar de ativação foi absorvido pelo somatório, $g(\boldsymbol{x})$, e, portanto, 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.
- 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.

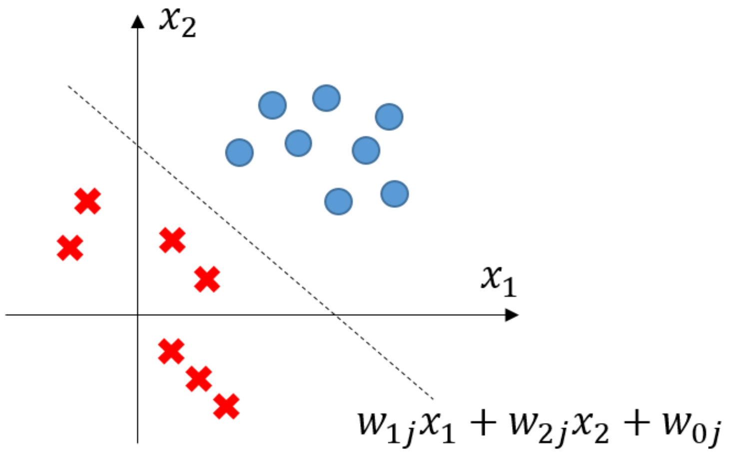
$$g(\boldsymbol{x}) = \sum_{i=0}^{N} w_i x_i = 0.$$





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*** 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 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 torno de 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:

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

Regra de aprendizado do perceptron

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

$$e = d - y.$$

- 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, geralmente, aplicada a um exemplo de entrada por vez. Os exemplos são escolhidos aleatóriamente, assim como o que é feito com o ***gradiente descendente estocástico***.

Regra de aprendizado do perceptron

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

- Percebam 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:
  1. Se a saída for correta, i.e., $d = y$, então os pesos não são atualizados.
  2. 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 $\boldsymbol{w}^T\boldsymbol{x}$ aumente tal que y se torne 1.
  3. Se d = 0 mas 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

```
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.

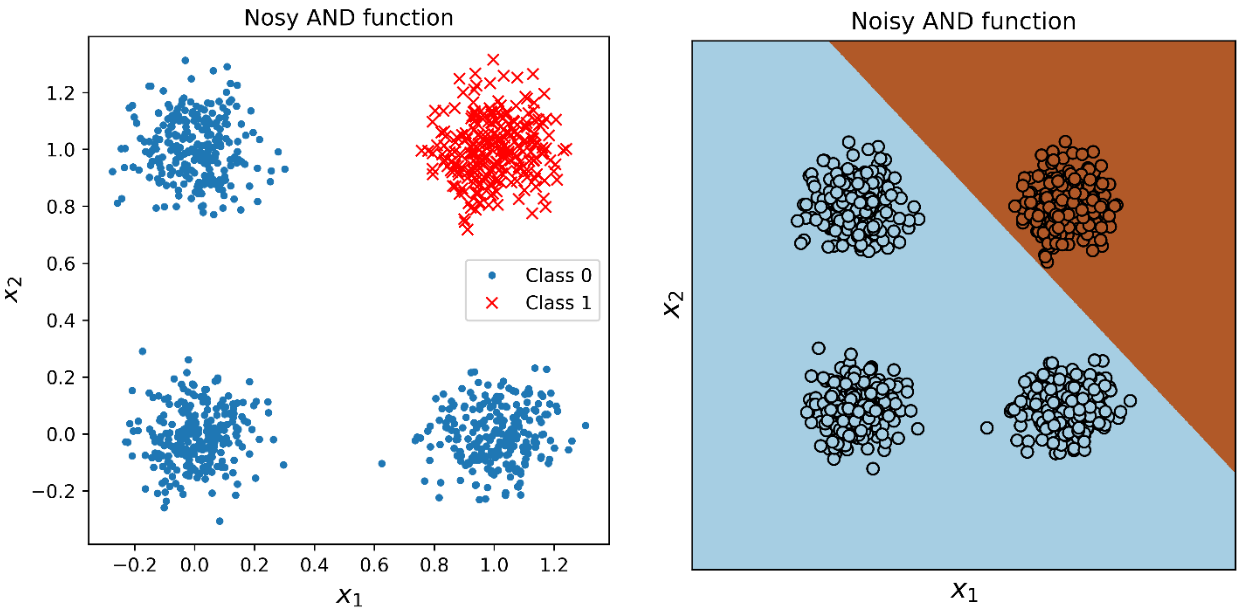
Adiciona ruído aos atributos de entrada

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.

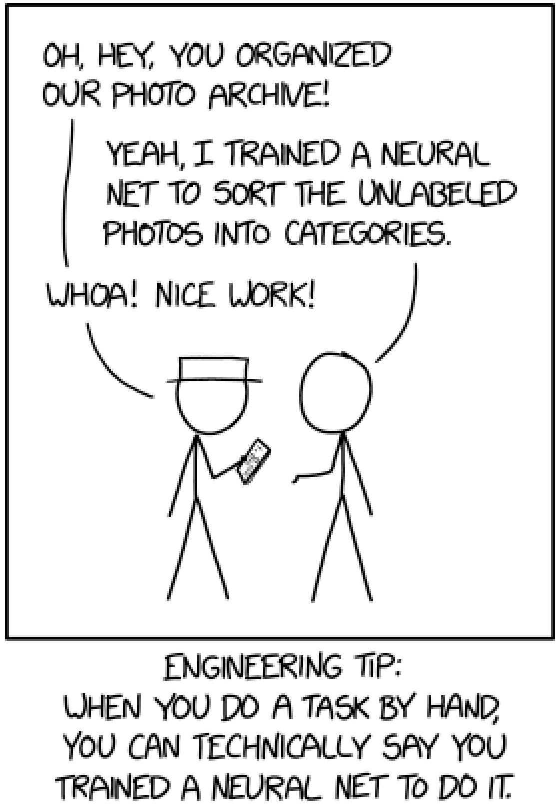
[Exemplo: Perceptron.ipynb]

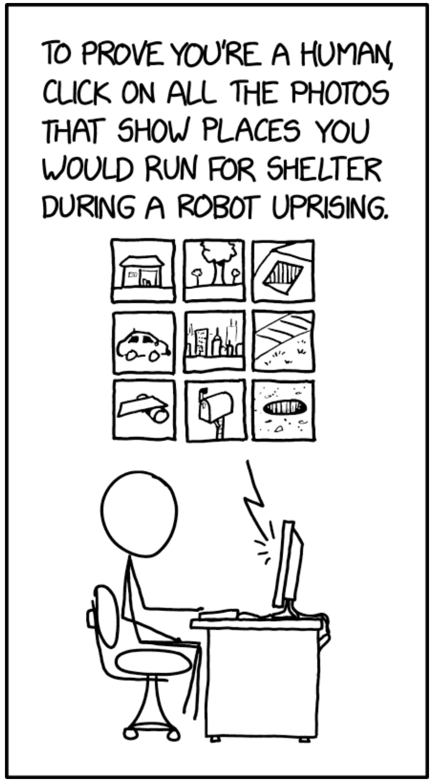
Avisos

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

Obrigado!


THE #1 deep learning expert EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY model's training"
HEY! GET BACK TO WORK!
training!
OH. CARRY ON.

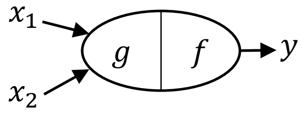

OH, HEY, YOU ORGANIZED OUR PHOTO ARCHIVE!
YEAH, I TRAINED A NEURAL NET TO SORT THE UNLABELED PHOTOS INTO CATEGORIES.
WHOA! NICE WORK!
ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

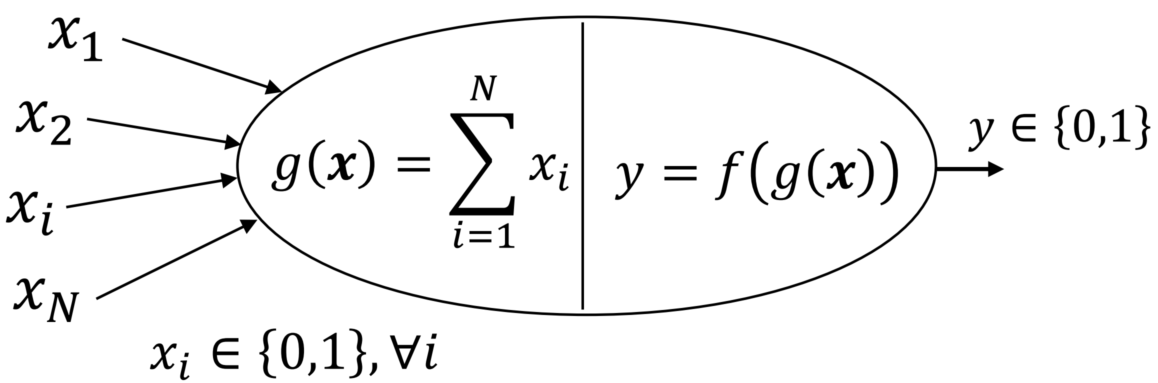

TO PROVE YOU'RE A HUMAN, CLICK ON ALL THE PHOTOS THAT SHOW PLACES YOU WOULD RUN FOR SHELTER DURING A ROBOT UPRISING.

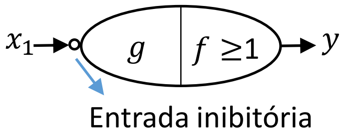

STILL WAITING
FOR MY NEURAL NETWORK TO TRAIN
memegenerator.net

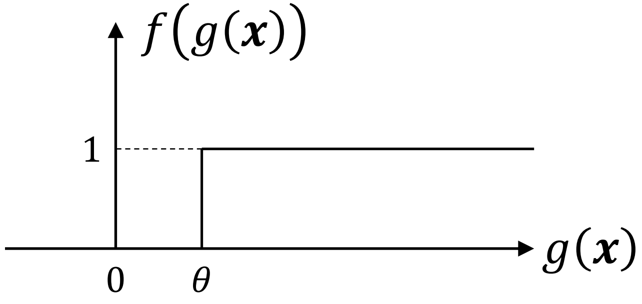

I DON'T USUALLY SPENT MY MONEY
BUT WHEN I DO I DUMP ALL OF THEM ON GPUS

Figuras



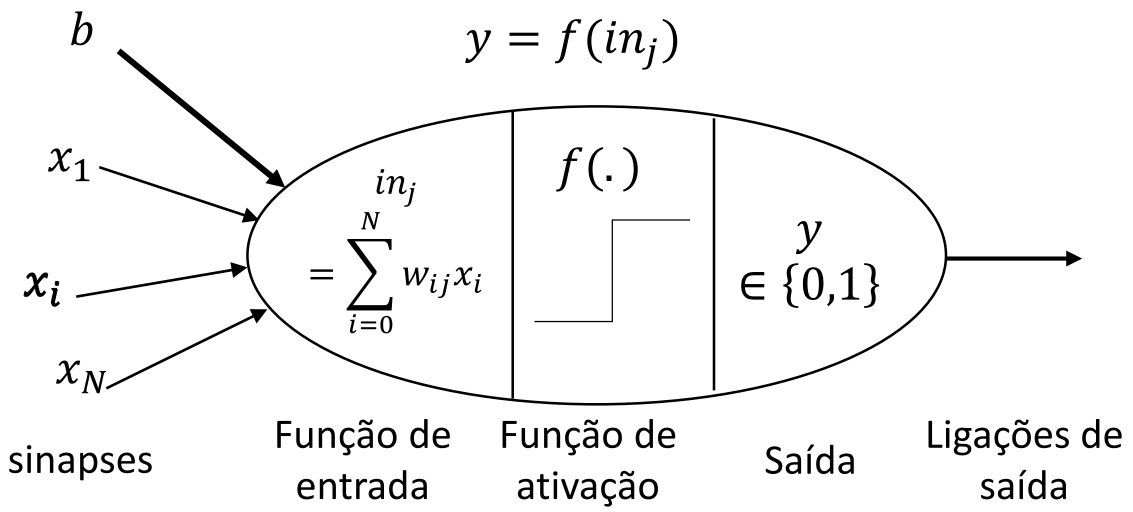




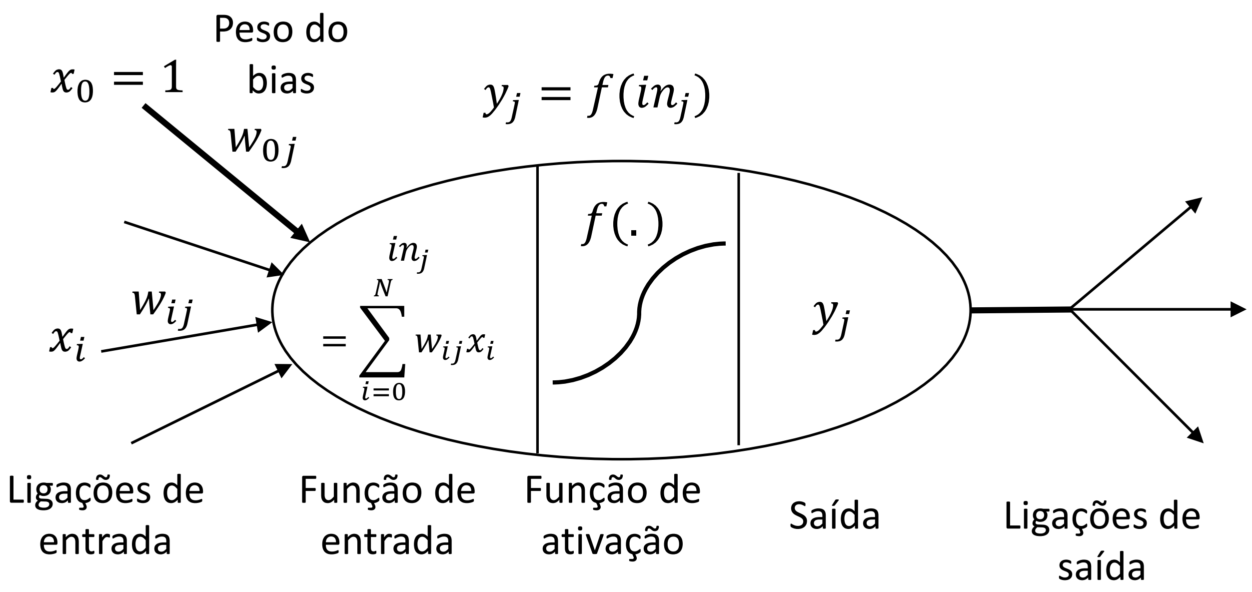


$$y = f\big(g(\boldsymbol{x})\big) = \begin{cases} 1 \text{ se } g(\boldsymbol{x}) \geq \theta \\ 0 \text{ se } g(\boldsymbol{x}) < \theta \end{cases}$$

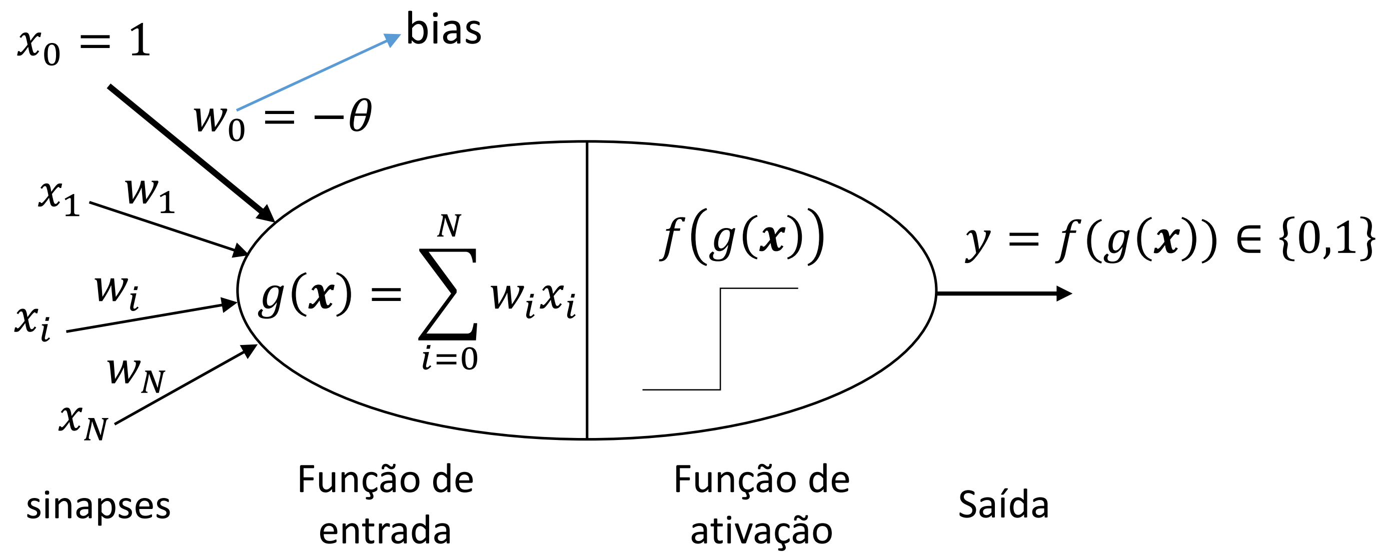
onde θ é o limiar de decisão.



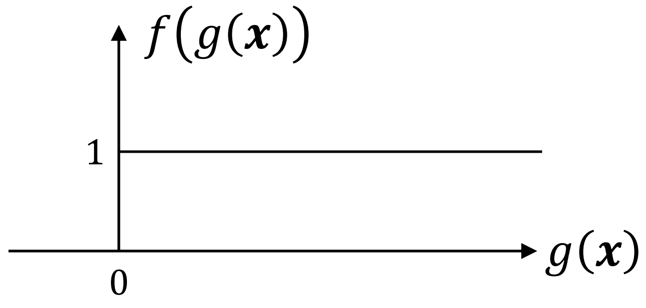
b
$y = f(in_j)$
x_1
x_i
x_N
$in_j = \sum_{i=0}^{N} w_{ij} x_i$
$f(.)$
$y \in \{0,1\}$
sinapses
Função de entrada
Função de ativação
Saída
Ligações de saída



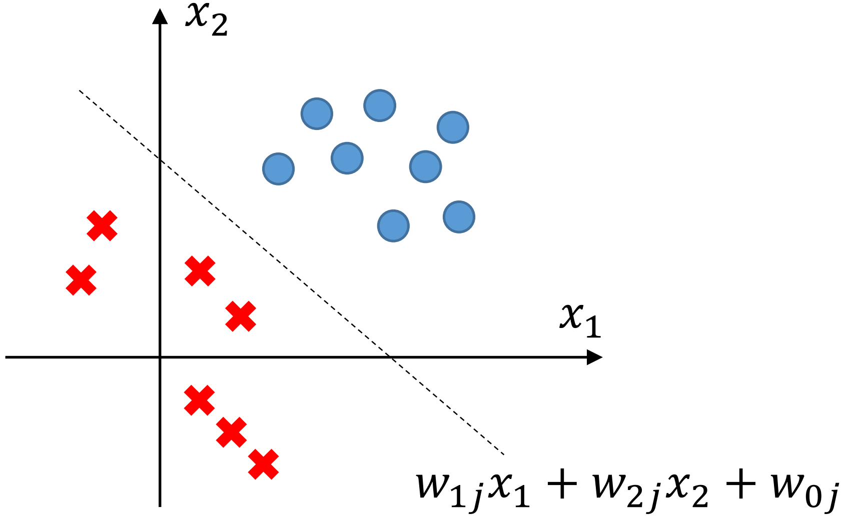
Peso do bias
$x_0 = 1$
w_{0j}
$y_j = f(in_j)$
w_{ij}
x_i
$in_j = \sum_{i=0}^{N} w_{ij} x_i$
$f(.)$
y_j
Ligações de entrada
Função de entrada
Função de ativação
Saída
Ligações de saída



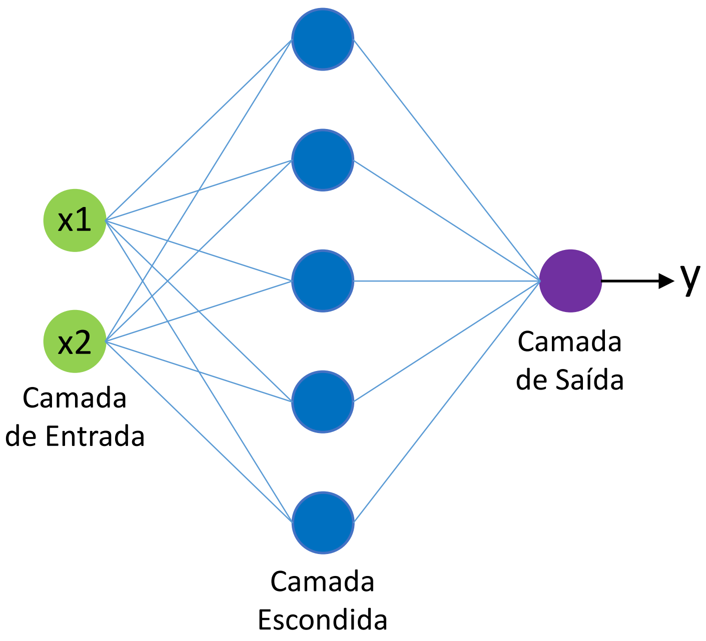
$x_0 = 1$
bias
$w_0 = -\theta$
x_1
w_1
w_i
x_i
w_N
x_N
$g(\boldsymbol{x}) = \sum_{i=0}^{N} w_i x_i$
$f(g(\boldsymbol{x}))$
$y = f(g(\boldsymbol{x})) \in \{0,1\}$
sinapses
Função de entrada
Função de ativação
Saída



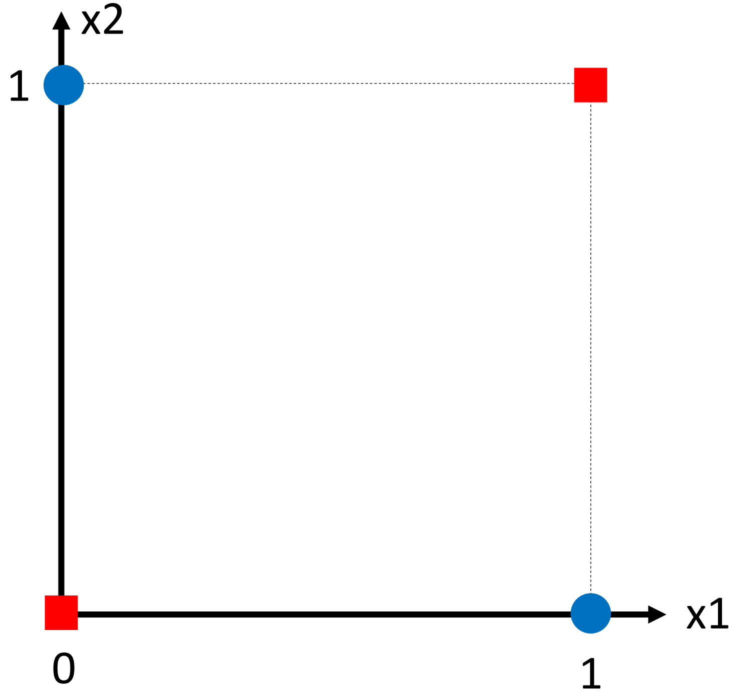
$f(g(\boldsymbol{x}))$
1
0
$g(\boldsymbol{x})$



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


x1
x2
y
Camada
de Entrada
Camada
Escondida
Camada
de Saída

XNOR



Classe 0 (nível lógico 0)

Classe 1 (nível lógico 1)