# Redes Neurais Artificiais

Professora: Anita Maria da Rocha Fernandes
Mestrando: Luiz Henrique A. Salazar

# Agenda

## Parte Teórica

- Introdução
- História das RNAs
- Conceitos Básicos

## Parte Prática

- Tensores
- Datasets

UNIVALI

# Parte Teórica

# Introdução

Redes Neurais Artificiais se fundamentam nos estudos sobre a estrutura do **cérebro humano** para tentar emular sua forma inteligente de processar informações.

# Introdução

Apesar de se desconhecer a maneira pela qual o cérebro manipula informações complexas, sabe-se que a **modelagem do conhecimento** contido em um problema específico pode ser representada através de **interconexões** entre células nervosas.

# Introdução

Estruturalmente, a rede neural artificial, também conhecida por **modelo conexionista** de computação, se assemelha à rede neural biológica pela composição de seus **neurônios** e pela **conexão** entre eles.

# Introdução

Redes Neurais Artificiais são **técnicas computacionais** que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem **conhecimento** através da **experiência**.

# Introdução

Uma grande rede neural artificial pode ter **centenas ou milhares** de unidades de processamento; já o cérebro de um mamífero pode ter **muitos bilhões** de neurônios.

## Introdução

Um modelo conexionista é uma estrutura de processamento de informações **distribuída** e **paralela**.

Ela é formada por unidades de processamento, comumente chamadas **neurônios**, interconectadas por arcos unidirecionais, também chamados de **conexões**.

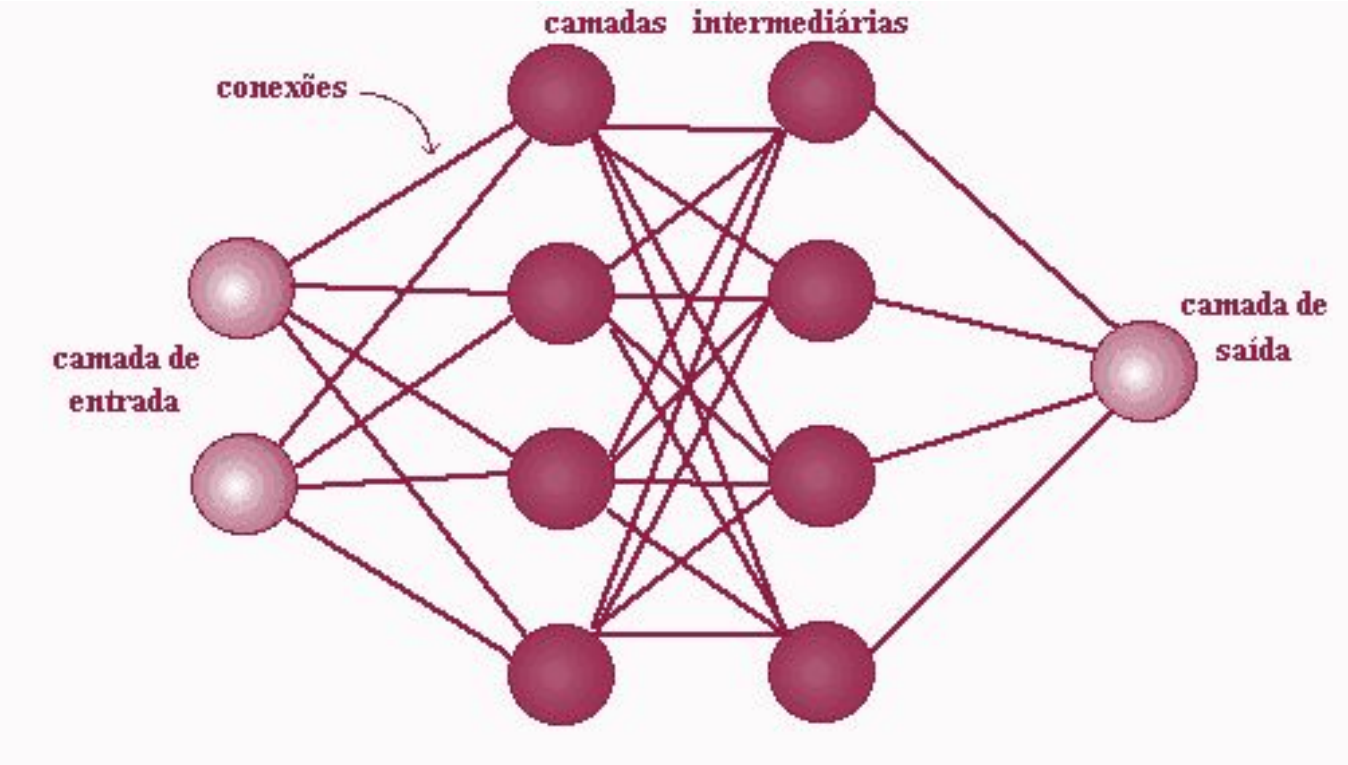
# Introdução

Todo o processamento que se realiza em cada unidade deve ser completamente **local**, isto é, deve depender apenas dos valores correntes dos **sinais de entrada** que chegam dos neurônios através das conexões.

## Introdução

Podemos dizer que as **RNAs** são sistemas **paralelos e distribuídos** compostos por unidades de processamento que calculam determinadas funções matemáticas, normalmente não lineares, dispostas em **uma ou várias camadas** que estão interligadas por diversas **conexões**, normalmente unidirecionais.

# Introdução


camadas intermediárias
conexões
camada de entrada
camada de saída

# Introdução

A capacidade das RNAs está associada a **capacidade de aprender** através de um conjunto de **exemplos**, fornecendo respostas coerentes para dados **desconhecidos**, o que a torna uma ferramenta computacional poderosa e eficaz na solução de problemas complexos.

## Introdução

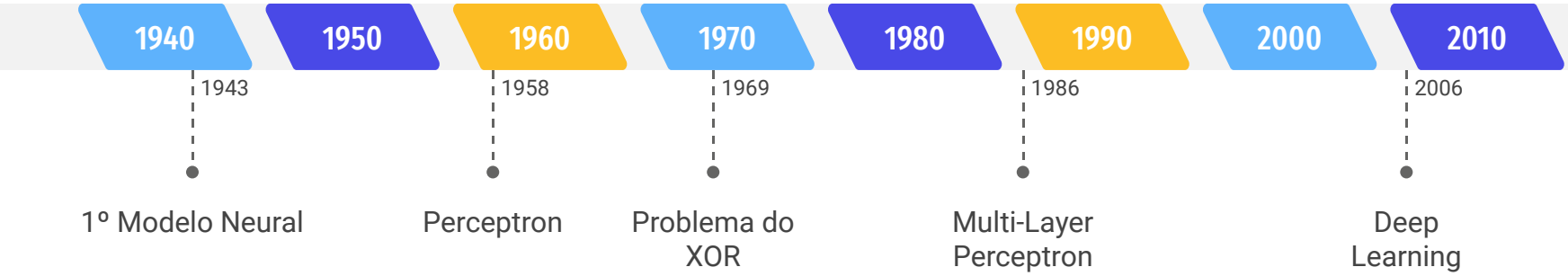
Um modelo de operação para as unidades de processamento pode ser resumido da seguinte maneira:

- Sinais são apresentados na **entrada**;
- Cada sinal é multiplicado por um **peso** que indica sua influência na saída;

# Introdução

- É realizada uma **soma** ponderada dos sinais que produz um **nível de atividade**; e
- Se este nível de atividade exceder certo **limite** a unidade de processamento produz uma determinada **resposta**.

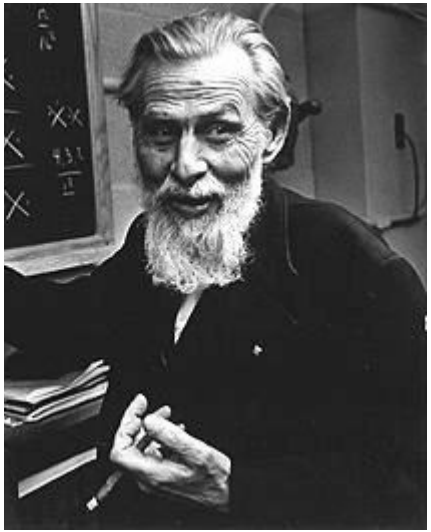
# História das Redes Neurais



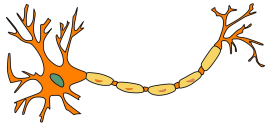
# Primeiro Modelo Neural

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

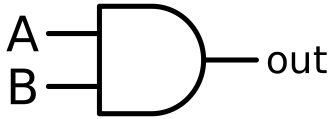
1943



Modelo computacional inspirado no **cérebro humano** de forma realista.



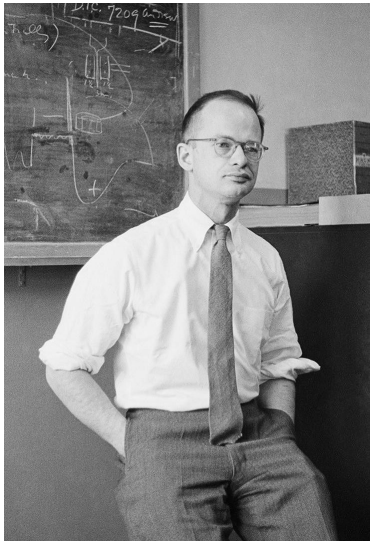
Analogia entre as células nervosas e o **processo eletrônico**.




**Warren Mcculloch**
Neurocientista

"A logical calculus of the ideas immanent in nervous activity" (1943)

**Walter Pitts**
Matemático



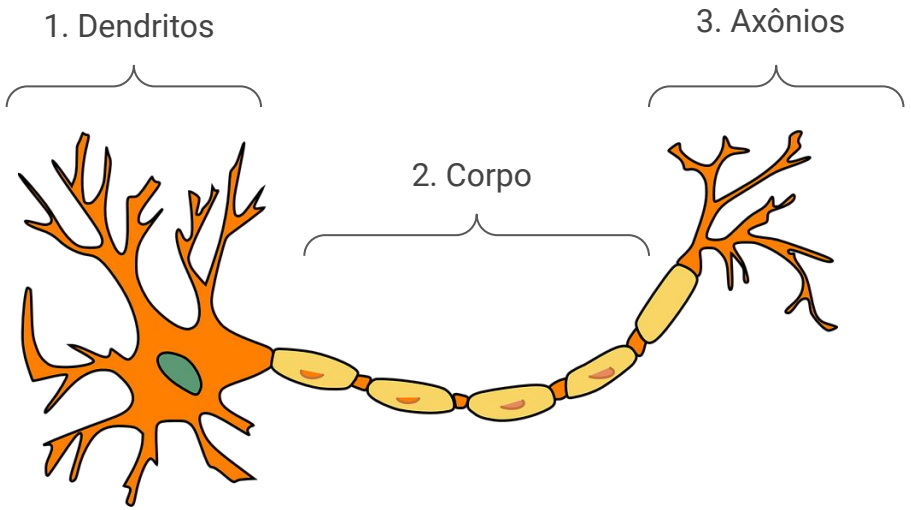
# Primeiro Modelo Neural

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1943

Redes neurais são inspiradas no cérebro humano. Sua unidade básica é o **neurônio**, composto por três partes principais:

1. **Dendritos**: Recebem sinais de outros neurônios.
2. **Corpo**: Processa os sinais recebidos do próprio neurônio.
3. **Axônio**: Transmite o sinal para outros neurônios. Sua ativação depende da força do sinal - pode ou não ativar, depende do estímulo que ele recebe.



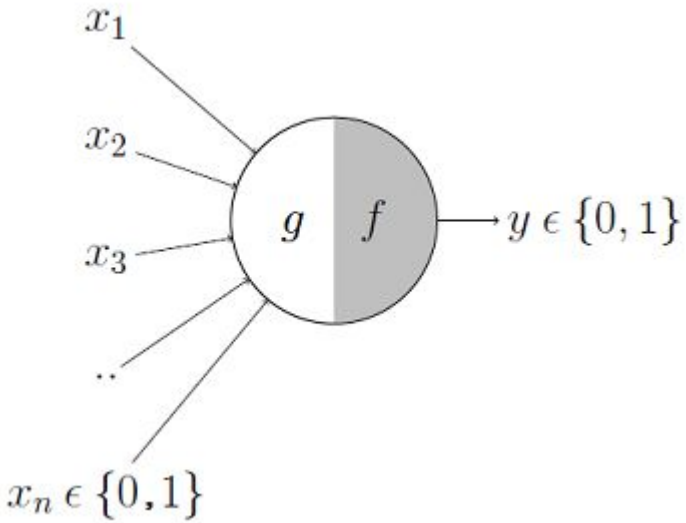
# Primeiro Modelo Neural

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1943

Primeira representação artificial do neurônio (modelo ***McCulloch-Pitts***):

- **Dendrito**: Recebe as entradas $x_1$, $x_2$, ..., $x_n$.
- **Corpo**: Função ***g*** agrega as entradas.
  - Agregação simples com somatório das entradas $x_1 + x_2 + \ldots + x_n$.
- **Axônio**: Função ***f*** decide a ativação do neurônio de acordo com a saída de ***g***.
  - *g*(x) > *limiar*; ativa o neurônio.



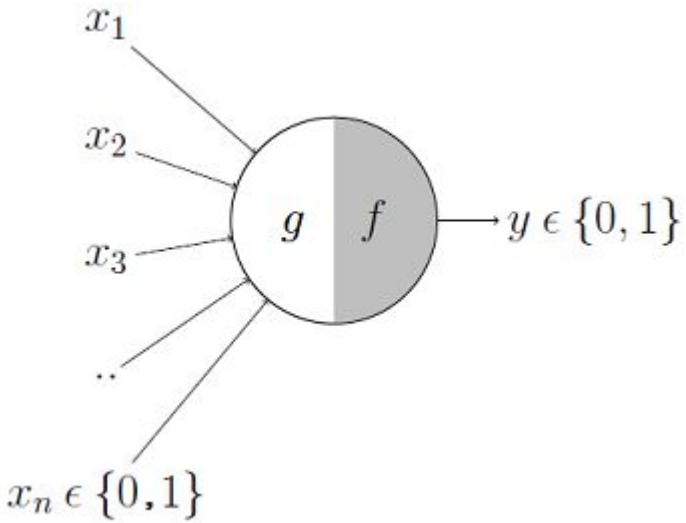
# Primeiro Modelo Neural

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1943

O neurônio de ***McCulloch-Pitts*** é binário, com poucos neurônios e sem técnica de retropropagação para ajustar os pesos.

Com o ajuste do limiar de ***f***, era possível realizar diferentes funções. Considerando que todas as entradas e saída são *booleanos*, os exemplos mais comuns são as funções *booleanas*:

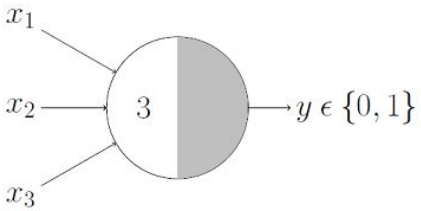


# Primeiro Modelo Neural

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

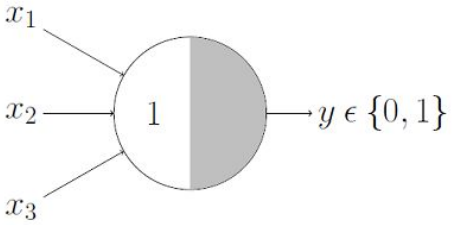
1943

- **AND**: ativa se a soma $g(x) \geq 3$. Todos os inputs iguais a 1.




---

- **OR**: ativa se a soma $g(x) \geq 1$, ou seja, $x_1 = 1$ **ou** $x_2 = 1$ **ou** $x_3 = 1$.



# Perceptron

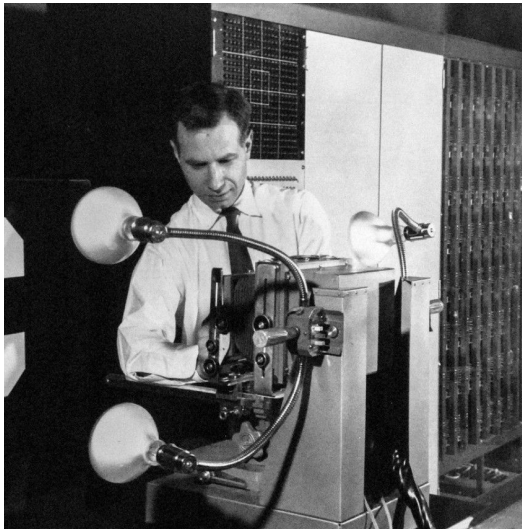
1940 | **1950** | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1958

Frank Rosenblatt propõe uma versão melhorada do modelo neural, em que associa-se um peso $w_i$ a cada entrada $x_i$. De acordo com o problema em questão, os pesos são **otimizados**.

Introduz o **aprendizado** nos modelos neurais através do **Perceptron**!

*"It's the **first machine** which is capable of **having an original idea**"*

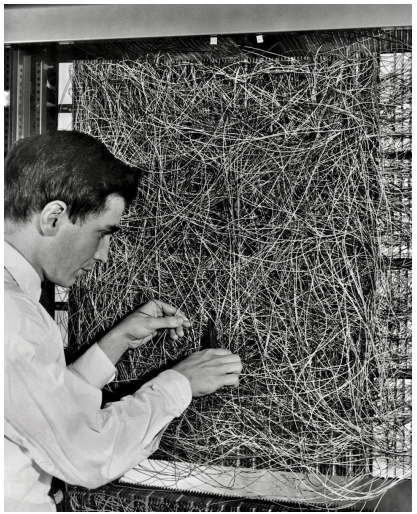


**Frank Rosenblatt**
Especialista em Psicologia Cognitiva

# Perceptron

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1958



Na prática, o modelo era um ***hardware*** que implementava uma rede de *perceptrons* para realizar a classificação de **padrões visuais**.

Cada conexão do neurônio era um **potenciômetro**, que permitia determinada tensão passar. Então, se o potenciômetro estivesse **mais solto** (passava mais tensão), ele dava um **peso maior** para aquela conexão.

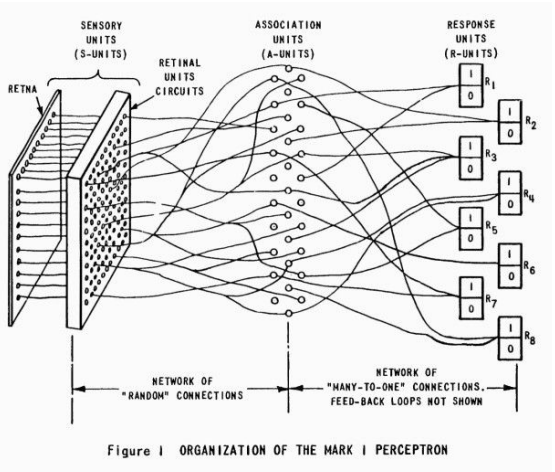


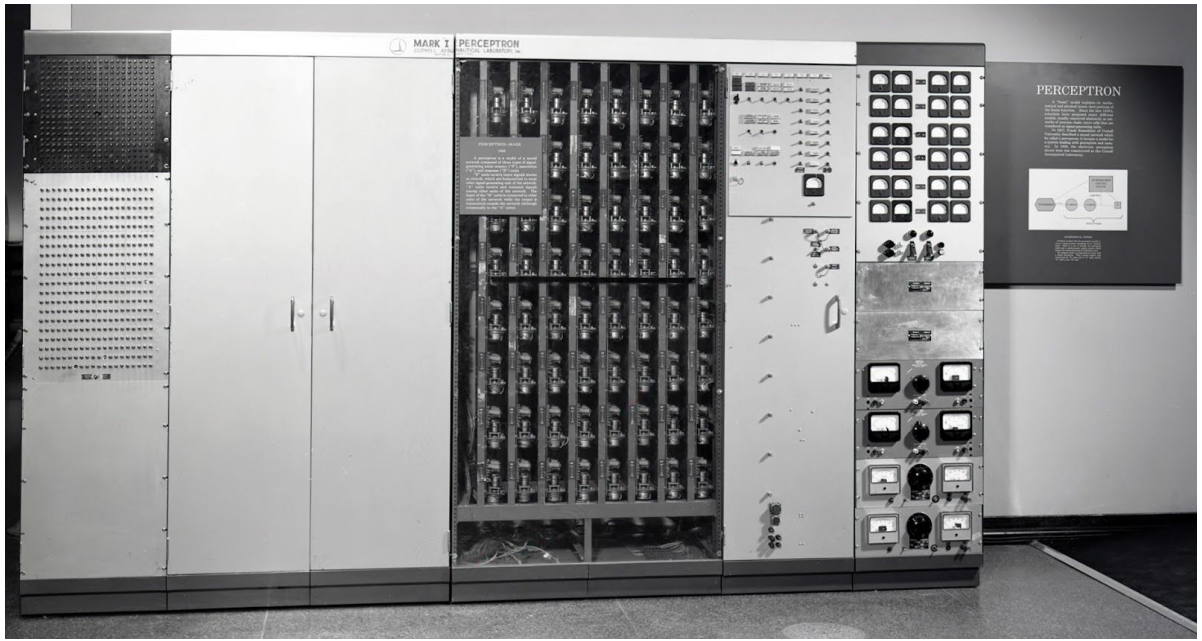
Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

# Perceptron

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1958

**Mark I Perceptron** em exposição no Smithsonian museum, Estados Unidos.



# Limitação do Perceptron

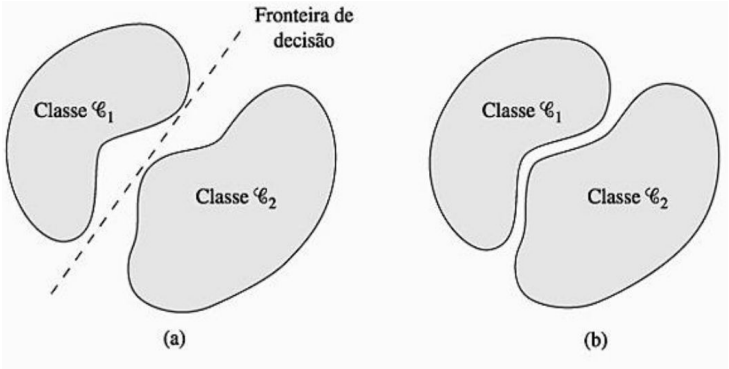
1940 | 1950 | **1960** | 1970 | 1980 | 1990 | 2000 | 2010

1969

Para o perceptron funcionar adequadamente, as duas **classes** devem ser ***linearmente separáveis***, ou seja:

**Perceptrons** só aprendem funções lineares - **retas**!

Logo, funções mais complexas (**não lineares**) não podem ser resolvidas com um perceptron.




(a) Par de padrões linearmente separáveis.
(b) Par de padrões não linearmente separáveis.

# Limitação do Perceptron

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1969

**Marvin Minsky**
Cientista Cognitivo

**Seymour Papert**
Cientista da Computação

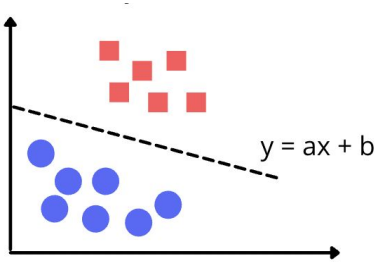
# Limitação do Perceptron

1940 | 1950 | **1960** | 1970 | 1980 | 1990 | 2000 | 2010

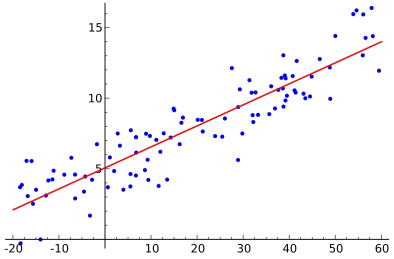
1969

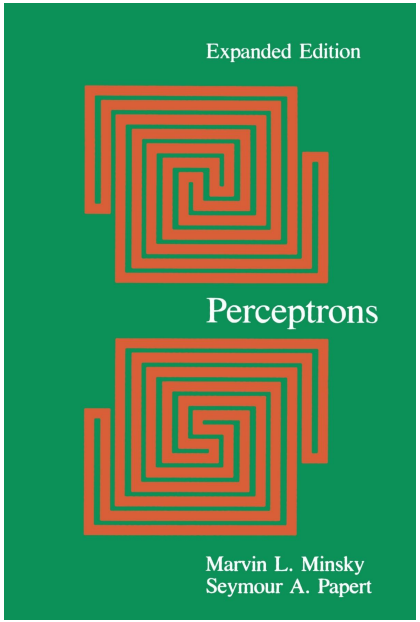
Escreveram o **livro** "*Perceptron*" em que são destacadas as já conhecidas limitações dos perceptron e das redes neurais.

Problemas de classificação **linear**:




Problemas de regressão **linear**:



# Problema do XOR

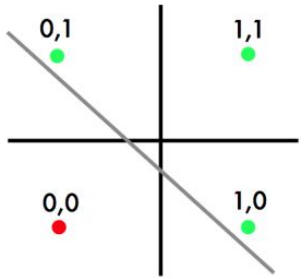
1940 | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010

1969

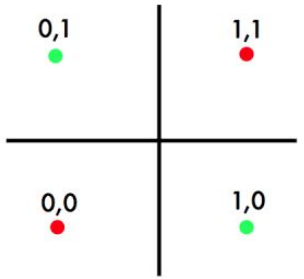
Operação lógica do **ou exclusivo (XOR)** representa a categoria de problemas que um perceptron não consegue resolver.

É **verdadeiro** quando os eventos **diferem** e são **falsos** quando os eventos são **iguais**.

Não existe reta que separa as duas classes.
**Função não-linear!**




OR




XOR

# Problema do XOR

1940 1950 1960 1970 1980 1990 2000 2010

1969

# Inverno da IA

1940 | 1950 | 1960 | **1970** | **1980** | 1990 | 2000 | 2010

1969 — 1986

Desde a popularização do livro do Minsky e do Papert, se iniciou o “**Inverno da IA”** que durou até 1986..

- Era de poucas novidades;
- Cortes nos investimentos e
- Baixa atenção à área.

# Multi-Layer Perceptron e Backpropagation

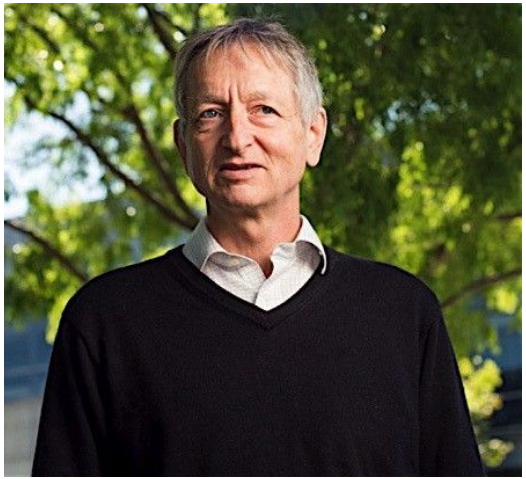
1940 | 1950 | 1960 | 1970 | **1980** | 1990 | 2000 | 2010

1986

Novo método de aprendizado: o **Backpropagation**.

Permite que rede com **múltiplas camadas** de perceptron (Multi-Layer Perceptron) aprendam funções mais complexas, como as não lineares.

Aprendizado **profundo** com múltiplas camadas.



**Geoffrey Hinton**
Cientista da Computação

# Multi-Layer Perceptron e Backpropagation

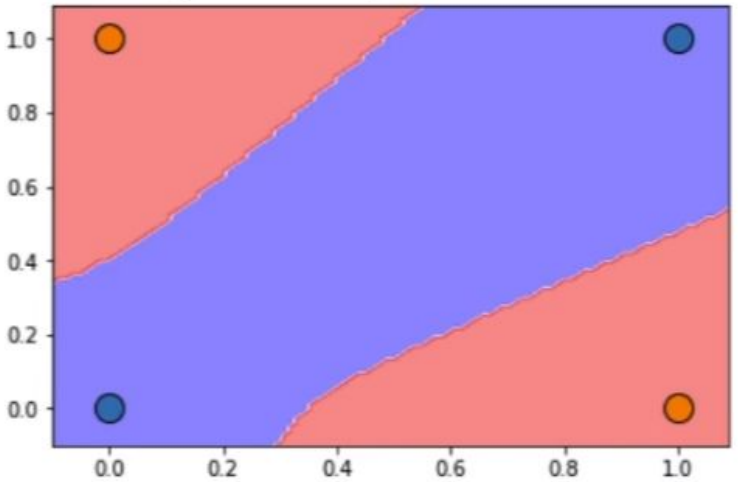
1940 | 1950 | 1960 | 1970 | **1980** | 1990 | 2000 | 2010

1986

Novo método de aprendizado: o **Backpropagation**.

Permite que rede com **múltiplas camadas** de perceptron (Multi-Layer Perceptron) aprendam funções mais complexas, como as não lineares.

Aprendizado **profundo** com múltiplas camadas.




**Solução do problema do XOR com MLP**

# Teorema da Aproximação Universal


Na mesma época foi postulado o **Teorema da Aproximação Universal.**

"Uma rede neural *feed forward** com apenas uma camada (escondida) é suficiente para **representar qualquer função**, mas a camada pode ser inviavelmente grande e pode falhar em aprender e generalizar corretamente."

Logo, **teoricamente**, se existe uma função também existe uma rede que a resolve.

** uma rede neural simples, da entrada para a saída*

# Teorema da Aproximação Universal


**Entretanto**.. naquela época **não era possível** validar o teorema na prática, devido a alguns problemas:

- Grande modelos neurais são **computacionalmente custosos**.
  - Não havia *hardwares* suficientes para processar.
- Um grande **volume de dados** é necessário para treinar os modelos.
  - Não haviam dados em abundância para treinar os modelos.

# *Deep Learning*

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006

Em 2006, surge o *deep learning* como algo **popular**, e já existia não só a **teoria** das redes neurais como também:

- **Hardware** robusto (GPUs)

# *Deep Learning*

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006

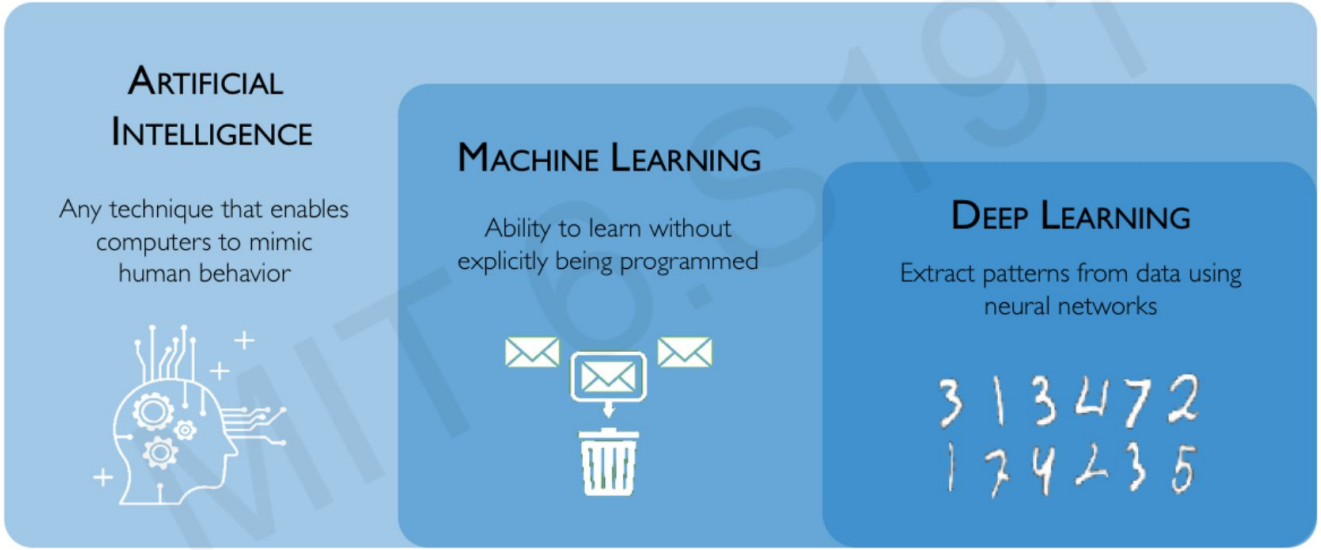
Em 2006, surge o *deep learning* como algo **popular**, e já existia não só a **teoria** das redes neurais como também:

- **Hardware** robusto (GPUs)
- Abundância de **dados**

# Deep Learning

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006



# *Deep Learning*

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006

Proporcionou o desenvolvimento de soluções que até poucas décadas atrás eram altamente complexas de serem solucionadas.

- **Classificação de Imagens**


*This network is running live in your browser

http://cs231n.stanford.edu/

# *Deep Learning*

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006

Proporcionou o desenvolvimento de soluções que até poucas décadas atrás eram altamente complexas de serem solucionadas.

- **Transferência de Estilo**

https://github.com/junyanz/CycleGAN

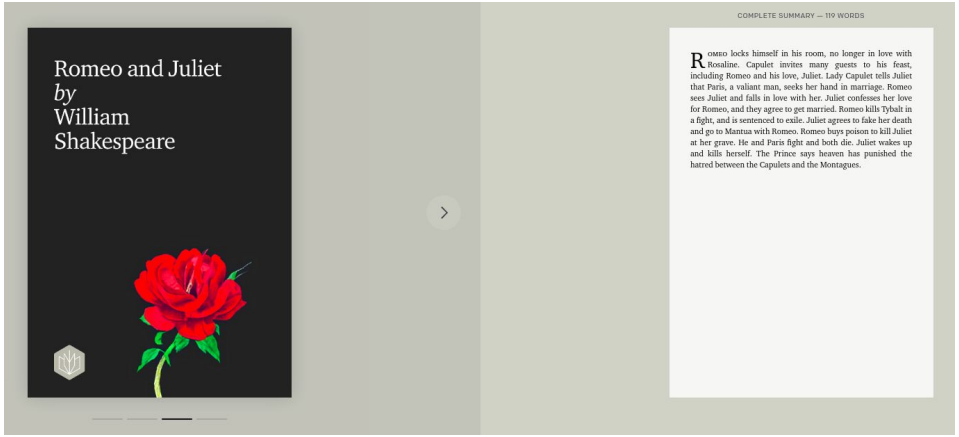
# Deep Learning

1940 | 1950 | 1960 | 1970 | 1980 | 1990 | **2000** | 2010

2006

Proporcionou o desenvolvimento de soluções que até poucas décadas atrás eram altamente complexas de serem solucionadas.

- **Modelos de Linguagem**




https://openai.com/blog/summarizing-books/

https://gpt3demo.com/

## Organização das RNAs em Camadas

Usualmente as camadas são classificadas em três grupos:

- **Camada de Entrada**: onde os padrões são apresentados à rede;
- **Camadas Intermediárias (Escondidas)**: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características;

## Organização das RNAs em Camadas

- **Camada de Saída**: onde o resultado final é concluído e apresentado.

A propriedade mais importante das redes neurais é a habilidade de **aprender** de seu **ambiente** e com isso **melhorar** seu desempenho.

## Organização das RNAs em Camadas

O aprendizado da rede é feito através de um **processo iterativo** de ajustes aplicados a seus pesos: o **treinamento**.

O aprendizado ocorre quando a rede neural atinge uma **solução generalizada** para uma classe de problemas.

## Organização das RNAs em Camadas

Denomina-se **algoritmo de aprendizado** a um conjunto de regras bem definidas para a solução de um problema de aprendizado.
Existem **muitos** tipos de algoritmos de aprendizado **específicos** para determinados modelos de RNAs

- Diferem entre si principalmente pelo modo como os **pesos são modificados**.

## Processos de Aprendizagem

Outro fator importante é a maneira pela qual uma rede neural se relaciona com o ambiente.

**Aprendizado Supervisionado**: quando é utilizado um agente externo que indica à rede a resposta desejada para o padrão de entrada.

# Processos de Aprendizagem


Aprendizado Supervisionado

## Processos de Aprendizagem

**Aprendizado Não Supervisionado** (auto-organização): quando não existe uma agente externo indicando a resposta desejada para os padrões de entrada.

**Reforço** (semi-supervisionado): quando um crítico externo avalia a resposta fornecida pela rede. Aplicado em robótica, por exemplo.

# Processos de Aprendizagem

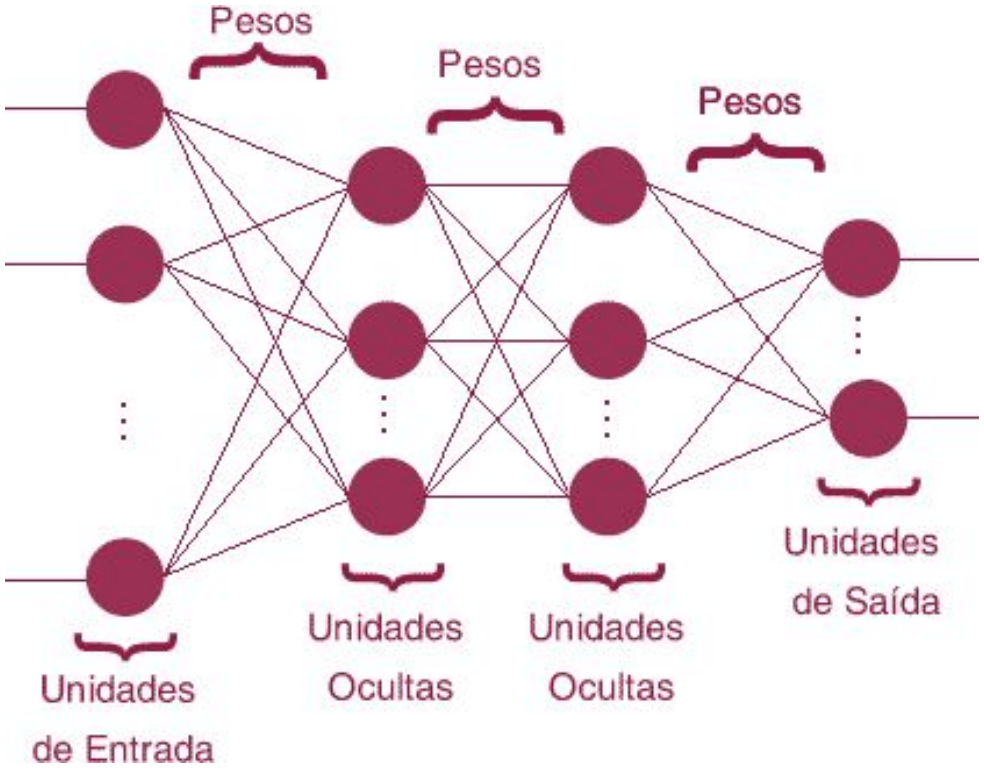

Aprendizado Não-supervisionado

## Multi-Layer Perceptron (MLP)

O PERCEPTRON Multicamadas é uma **extensão do PERCEPTRON** de camada única.

Esta arquitetura apresenta uma camada com unidades de **entrada**, conectada a uma ou mais unidades **intermediárias**, chamadas camadas ocultas, e uma camada de **saída**.

# Multi-Layer Perceptron (MLP)



## Multi-Layer Perceptron (MLP)

Utiliza o processo de **Aprendizagem Supervisionado**, sendo mais comum a utilização do algoritmo **Backpropagation**.

A rede possui **alto grau de conectividade**, o que permite interação entre as unidades.

## Multi-Layer Perceptron (MLP) - **Modelagem**

A modelagem da arquitetura de uma MLP envolve a escolha da **quantidade de camadas** e o **número de unidades** em cada camada.

- Escolha do número de unidades de **entrada**;
- Definição da **função de ativação** que irá ditar o comportamento da rede.

## Multi-Layer Perceptron (MLP) - **Treinamento**

Outros parâmetros devem ser escolhidos referentes ao treinamento:

- **taxa de aprendizado**
  - Indica a que **ritmo** os pesos são atualizados. Pode ser fixado ou alterado de modo adaptativo. O método atual mais popular é chamado Adam, o qual é um método que adapta a taxa de aprendizado.

## Multi-Layer Perceptron (MLP) - **Treinamento**

Outros parâmetros devem ser escolhidos referentes ao treinamento:

- **conjunto** de treinamento
  - Dados relevantes que destaquem as **características** que devem realmente ser aprendidas pela rede.

## Multi-Layer Perceptron (MLP) - **Camadas**

O processamento de cada unidade é influenciado pelo processamento efetuado pelas unidades das **camadas anteriores**.

Cada camada desempenha um papel específico dentro da rede.

## Multi-Layer Perceptron (MLP) - **Camadas**

Camada de **Entrada**:

- Receptora de estímulos

Camada de **Saída**:

- Combina as regiões formadas pelas camada anteriores, definindo o espaço de saída da rede.

## Multi-Layer Perceptron (MLP) - **Camadas**

Camadas **Intermediárias** (Ocultas):

- Atuam como **detectores de características**, que são representadas, internamente, através dos de pesos sinápticos.
- **Número** ideal de camadas intermediárias:
  - **Uma** camada é suficiente para aproximar qualquer função **contínua** e **duas** camadas  para aproximar qualquer **função matemática**.

## Multi-Layer Perceptron (MLP) - **Unidade**

Escolha do **número de unidades** em cada camada:

- O número de **exemplos** de treinamento
- A **complexidade** da função a ser aprendida pela rede;
- A **distribuição estatística** dos dados de treinamento.

## Multi-Layer Perceptron (MLP) - **Unidade**

Cuidados com o número de unidades por camada:

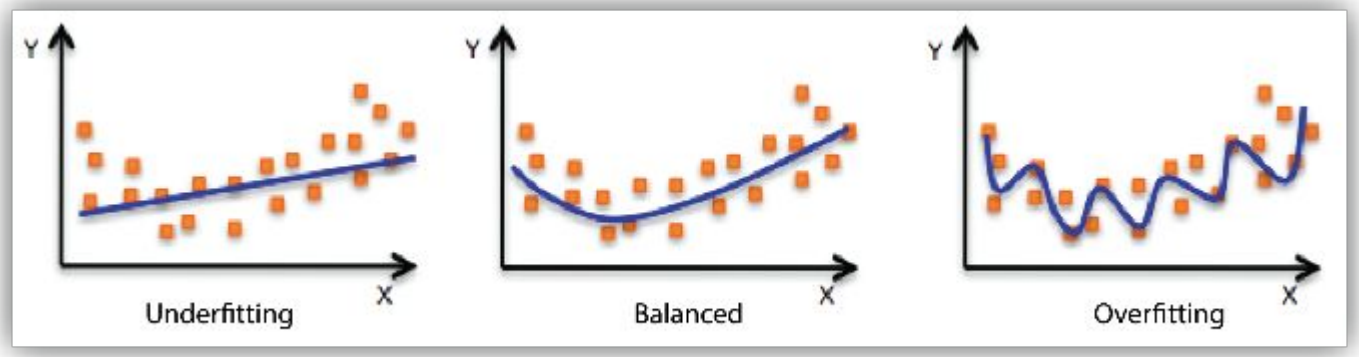
- Número **alto** de unidades pode fazer com que a rede **memorize** os dados do **treinamento**!
  - Torna-se incapaz de **generalizar** e, portanto, reconhecer **padrões não vistos** durante o treinamento.
  - **OVERFITTING!**

## Multi-Layer Perceptron (MLP) - **Unidade**

Cuidados com o número de unidades por camada:

- Número de unidades muito **pequeno**
  - A rede pode gastar muito tempo para aprender, podendo **não alcançar os pesos adequados**, ou seja, generalizar demais os padrões de entrada.
  - **UNDERFITTING!**

# Multi-Layer Perceptron (MLP) - **Unidade**



# Funções de Ativação

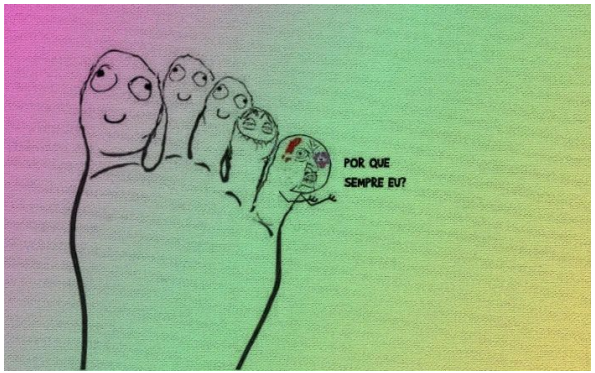
De forma abstrata, um neurônio interpreta as suas entradas e libera uma **ativação** com determinada força.

- Neurônio **ativa** quando ouvimos uma boa música

# Funções de Ativação

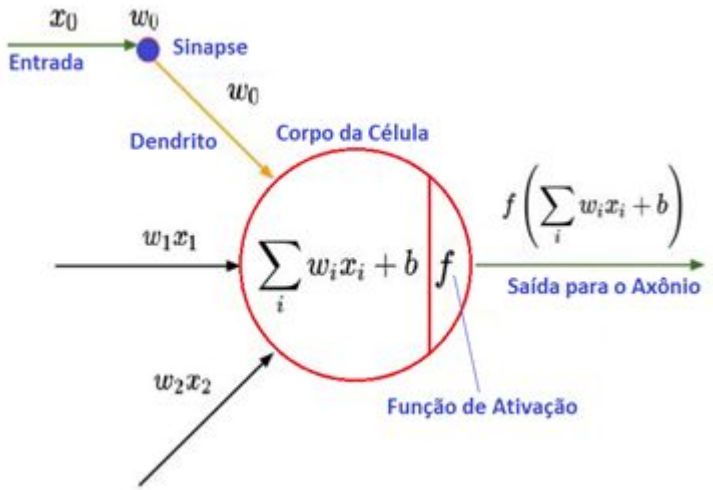
De forma abstrata, um neurônio interpreta as suas entradas e libera uma **ativação** com determinada força.

- Neurônio (provavelmente) **não ativa** quando batemos o dedão na quina da mesa :(



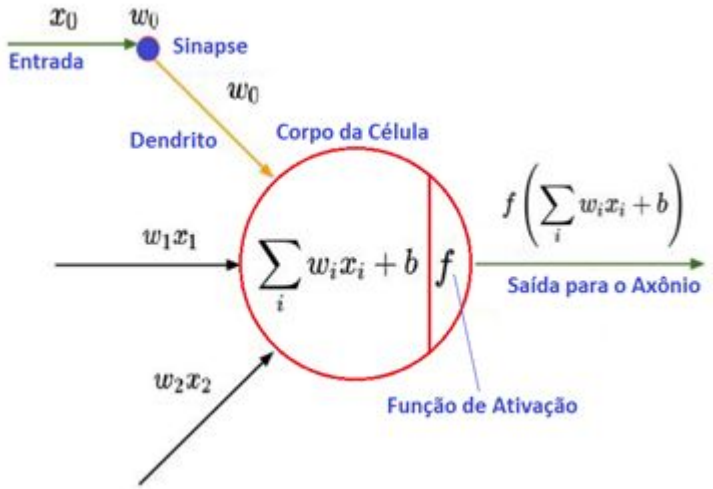
# Funções de Ativação

Em termos práticos, um neurônio deve **ativar** quando suas **entradas** apresentam o **padrão** para o qual ele foi **treinado.**



# Funções de Ativação

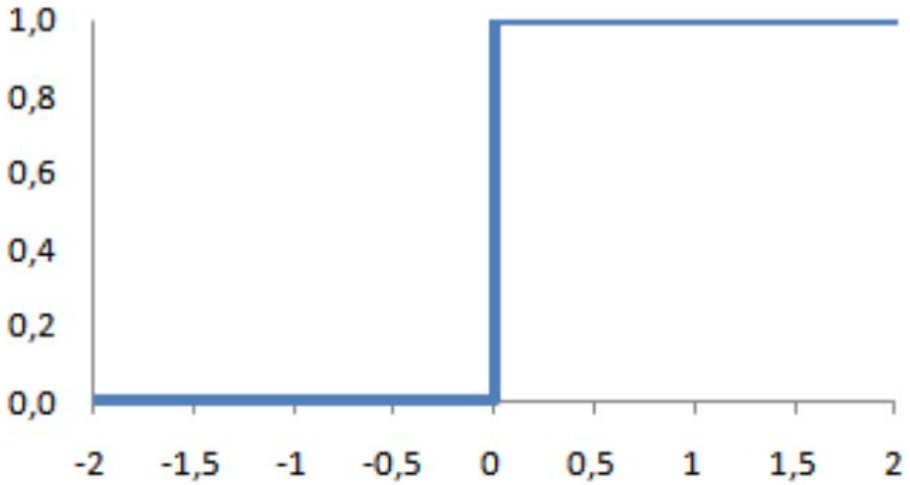
O papel da **função de ativação** é definir se o neurônio vai **ativar** e qual a sua **força** de ativação.



# Funções de Ativação - Função Limiar (Degrau)

Utilizada no modelo de McCulloch e Pitts, a função limiar modela a característica **"tudo-ou-nada"** deste neurônio.

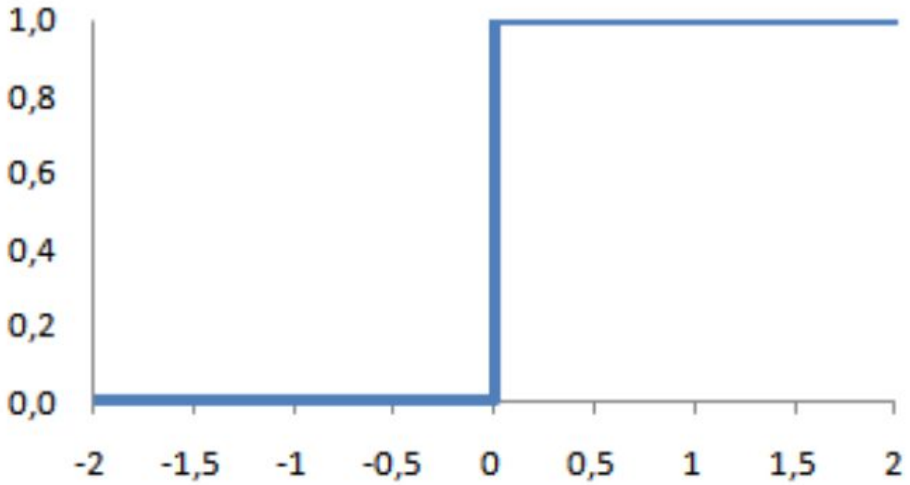
$$f(v) = \begin{cases} 1, \text{ se } v \geq 0; \\ 0, \text{ se } v \leq 0; \end{cases}$$



# Funções de Ativação - Função Limiar (Degrau)

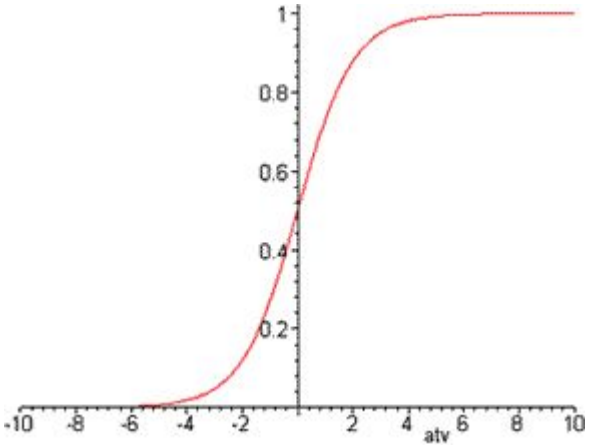
É uma transformação **não linear** que limita a saída do perceptron ao **intervalo {0, 1}**, como um interruptor.

$$f(v) = \begin{cases} 1, \text{ se } v \geq 0; \\ 0, \text{ se } v \leq 0; \end{cases}$$



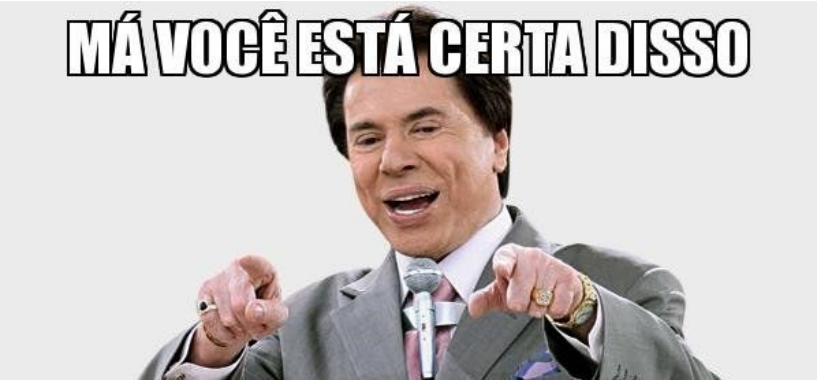
# Funções de Ativação - Função Sigmóide

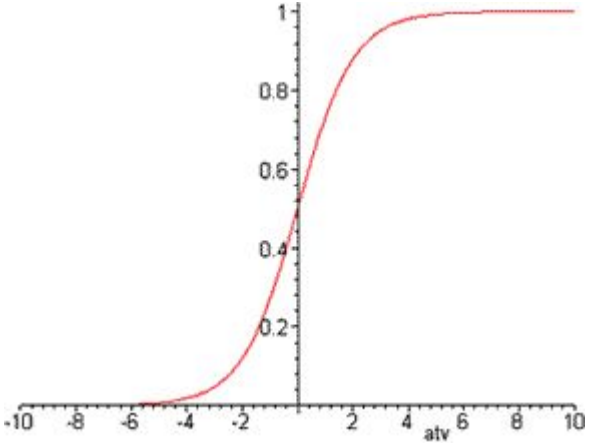
Para que o neurônio possa ativar com diferentes forças, existem funções como a **sigmóide**. Ela define um **intervalo [0, 1]**, mas pode assumir qualquer valor dentro desse intervalo.



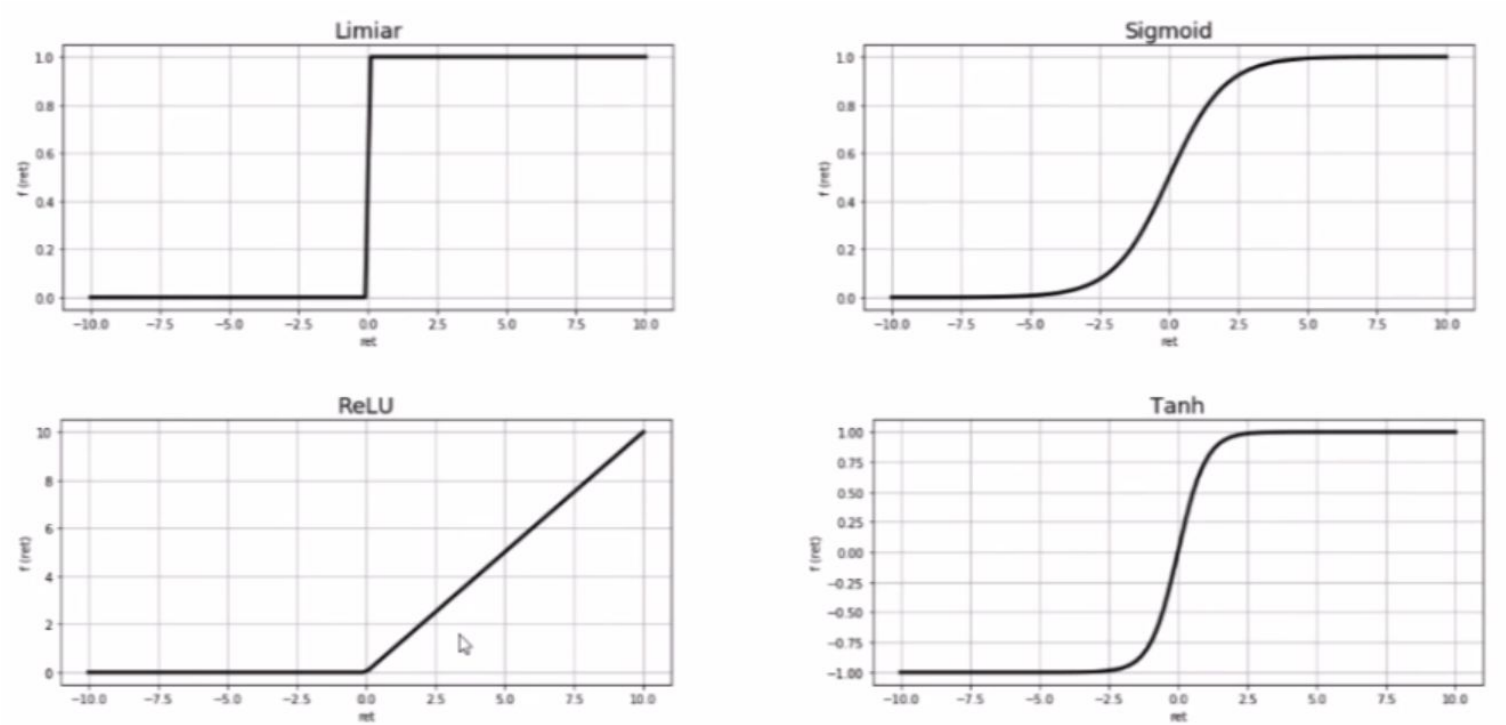
# Funções de Ativação - Função Sigmóide

Funções como a sigmóide permitem que se saiba o grau de **“certeza”** do perceptron.



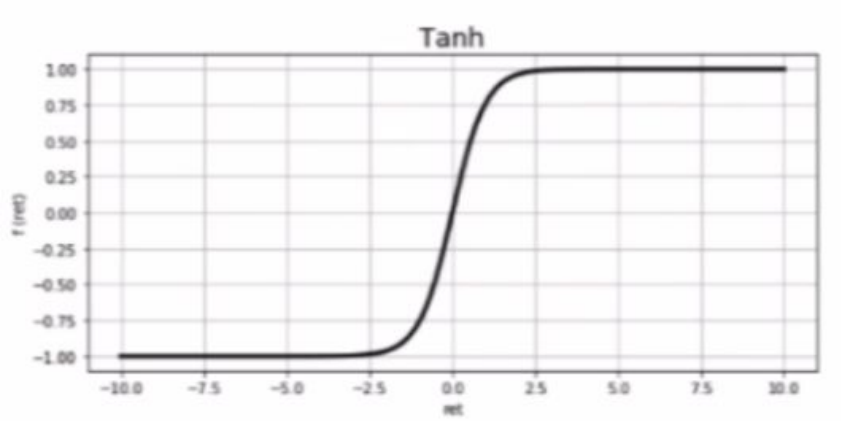
# Funções de Ativação - Funções mais populares



# Funções de Ativação - Funções mais populares

## Função Tangente Hiperbólica:

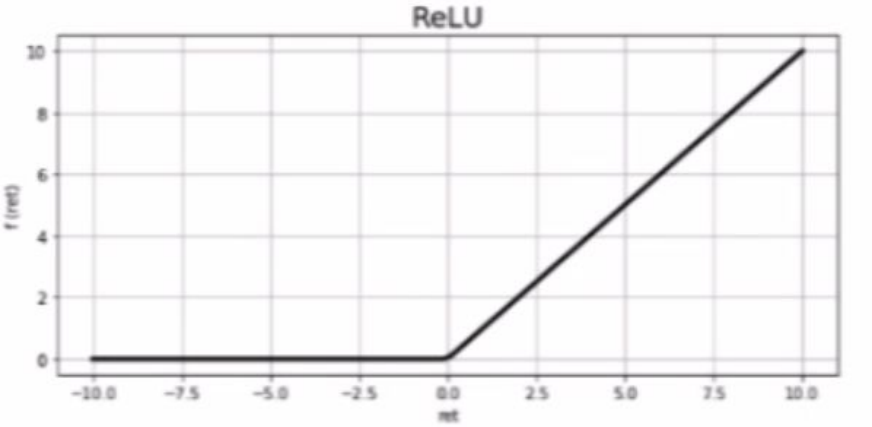
Como a sigmóide, também possui forma de “s”, porém assume valores entre **1 e -1**.



# Funções de Ativação - Funções mais populares

## Função ReLu:

É **linear** para valores **positivos** e simplesmente corta valores negativos os transformando em 0.



## Funções de Ativação - Funções mais populares

**Função Softmax:**

É um tipo de função sigmóide, mas é útil quando tentamos lidar com problemas de **classificação** com mais de uma classe.

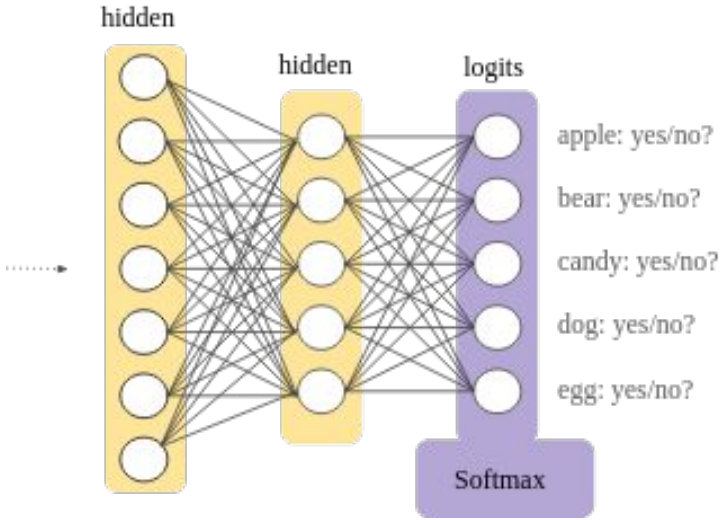
## Funções de Ativação - Funções mais populares

**Função Softmax:**
Transforma as saídas para cada classe para valores entre 0 e 1 e também **divide pela soma** das saídas.

Essencialmente, a função softmax dá a **probabilidade** de a entrada estar em uma determinada classe.

# Funções de Ativação - Funções mais populares

## Função Softmax:




| Class | Probability |
|---|---|
| apple | 0.001 |
| bear | 0.04 |
| candy | 0.008 |
| dog | 0.95 |
| egg | 0.001 |

# Funções de Ativação

No geral, as redes neurais (com mais de uma camada) são capazes de **solucionar problemas** tão **complexos** como geração de imagens artificiais ou modelos de linguagem graças às **funções de ativação**.

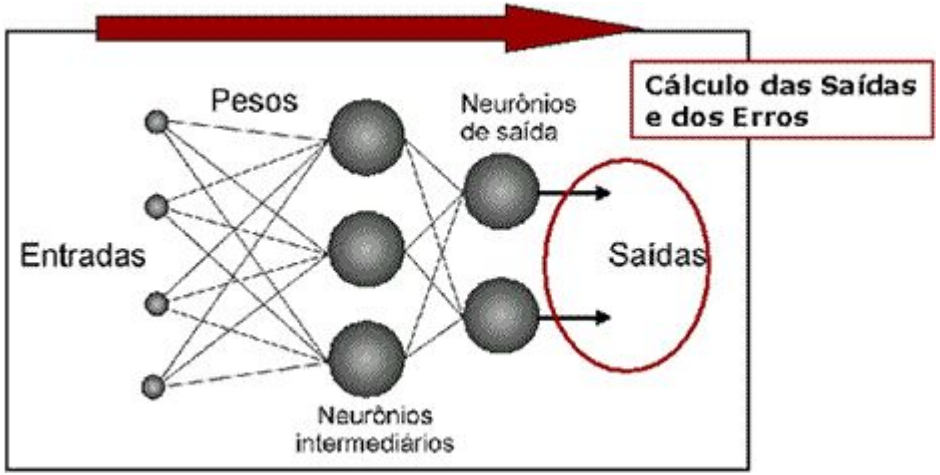
## Backpropagation

É o **algoritmo** para treinamento de Redes Multi-Camadas mais difundido.
Baseia-se no Aprendizado **Supervisionado** por Correção de Erros, constituído de:

- Propagação e
- Retropropagação.

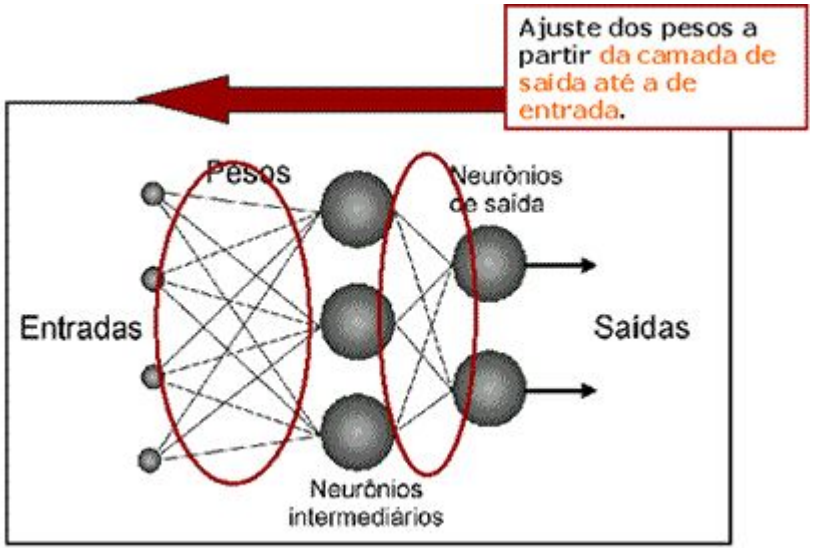
# Backpropagation - Propagação

Depois de apresentado o padrão de entrada, a resposta de uma unidade é **propagada** como entrada para as unidades na **camada seguinte**, até a camada de saída, onde é obtida a **resposta** da rede e o **erro** é calculado.



# Backpropagation - Retropropagação

Desde a camada de saída até a camada de entrada, são feitas alterações nos pesos sinápticos.



## Backpropagation

Durante a fase treinamento deve-se apresentar um conjunto formado pelo par: **entrada** para a rede e **valor desejado** para resposta a entrada.

A **saída** será comparada ao **valor desejado** e será computado o **erro** global da rede, que influenciará na **correção dos pesos** no passo de retropropagação.

# Backpropagation

Apesar de não haver garantias que a rede forneça uma **solução ótima** para o problema, este processo é muito utilizado por apresentar uma **boa solução** para o treinamento de Perceptrons Multi-Camadas.

# Parte Prática

# Frameworks

Os **frameworks** especializados em *deep learning* facilitam muito a **implementação** de uma rede neural.



## Frameworks

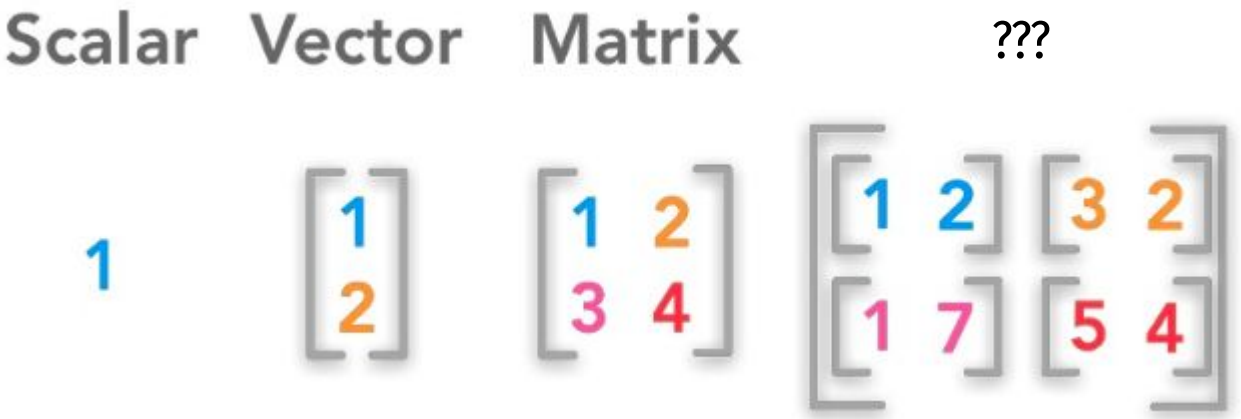
A estrutura de dados utilizada por esses frameworks é o **tensor**. As entradas, saídas e transformações de uma rede neural são representadas através dessa estrutura de dedos.




# Tensores

Tensores podem ser interpretados como uma generalização de estruturas já conhecidas.

- Estrutura com **3 dimensões**



# Tensores

Tensores podem ser interpretados como uma generalização de estruturas já conhecidas.

# Tensores

Na programação, quando a quantidade de dimensões é maior que 2, consideramos um ***nd-array***, ou seja, um array de n dimensões

```
# Escalar
escalar = 15

# Vetor / Array
array = [15, 16, 17]

# Matriz / Array 2D (2 x 3)
matriz = [[1, 2, 3],
          [4, 5, 6]]

# Array 3D (3 x 2 x 2)
array_3d = [ [[1, 2], [3, 4]],
             [[5, 6], [7, 8]],
             [[9, 10], [11, 12]] ]
```

Na matemática, os *nd-arrays* são chamados de **tensores** ou *nd-tensores*.

Logo, um tensor nada mais é do que um **array *n-dimensional***.

# Tensores

| Computer Science | Mathematics |
| --- | --- |
| (0D)number | Scalar |
| (1D) Array | Vector |
| (2D) Array | Matrix |
| (n D) Array | Tensor |

# Tensores

Por convenção, nos frameworks as estruturas são chamadas de **tensor** independente da dimensão.

| Dimensões | **Computação** | Frameworks |
|---|---|---|
| 0 | número | tensor 0d |
| 1 | array | tensor 1d |
| 2 | array 2d | tensor 2d |
| 3 | array 3d | tensor 3d |
| ... | ... | ... |
| n | array nd | tensor nd |

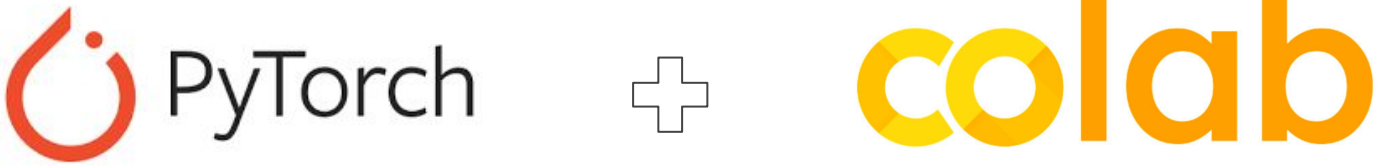
## Tensores

Em *deep learning*, é comum trabalhar com informação em alta dimensionalidade. Assim, o tensor é a **estrela do *deep learning***, inclusive nomeando um dos *frameworks* mais famosos da atualidade.

TensorFlow

# PyTorch

Desenvolvido pelo grupo de pesquisa de IA do **Facebook** e de código aberto no GitHub em 2017. Pytorch tem uma reputação de **simplicidade**, **facilidade** de uso, flexibilidade, uso eficiente de memória e gráficos computacionais dinâmicos.



# Datasets

kaggle

# Datasets

- Titanic: passageiro sobreviveu ao desastre?

https://www.kaggle.com/jamesleslie/titanic-neural-network-for-beginners

- Classificador de câncer de mama (duas classes - binário): https://archive.ics.uci.edu/ml/datasets/Breast+Cancer
- Classificador de flores (múltiplas classes): https://archive.ics.uci.edu/ml/datasets/Iris
- Classificador de região dos vinhos: https://archive.ics.uci.edu/ml/datasets/Wine