

# Redes Neurais

# Aprendizagem Simbólica versus Neuronal

## Aprendizagem Simbólica

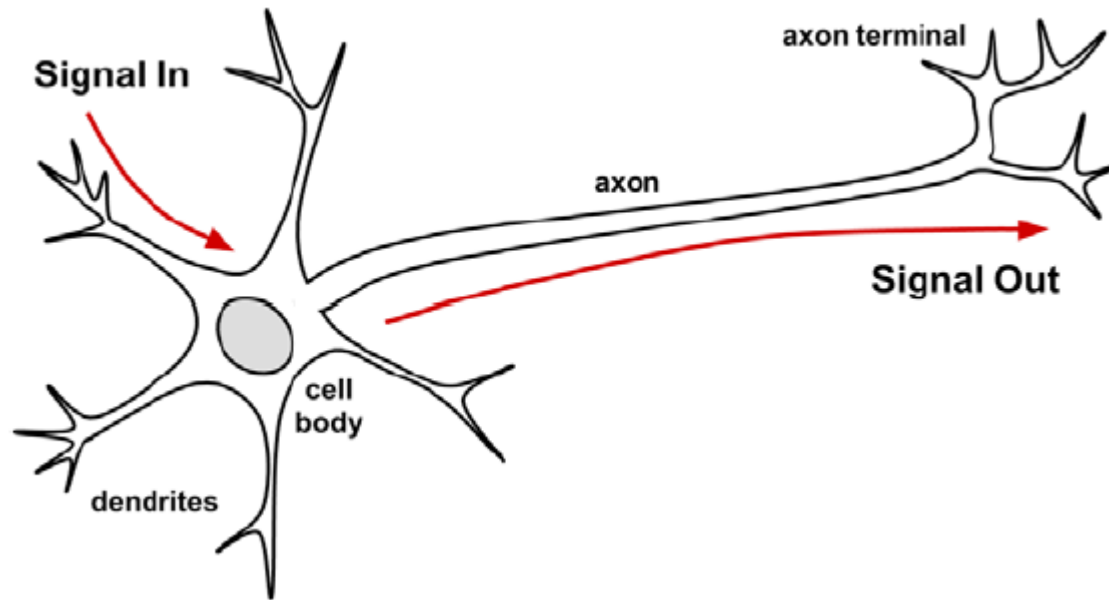
- Indução de Regras e Árvores de decisão
- trabalha com combinações discretas de valores de atributos
- usa operadores lógico/relacionais ( $=$ ,  $>$ ,  $<$ )

## Aprendizagem Neuronal

- trabalha ajustando pesos não-lineares e contínuos das suas entradas
- usa operadores numéricos ( $\times$ ,  $+$ )
- faz uma busca num espaço de granularidade mais fino do que os algoritmos de indução de regras

# Redes Neurais

Inspiradas no cérebro humano que consiste num enorme número de neurónios com altíssima inter-conectividade



São constituídas por uma série de nós (ou neurónios) interligados (através de conexões com pesos numéricos) arranjados em níveis

# Modelo Matemático do Neurónio

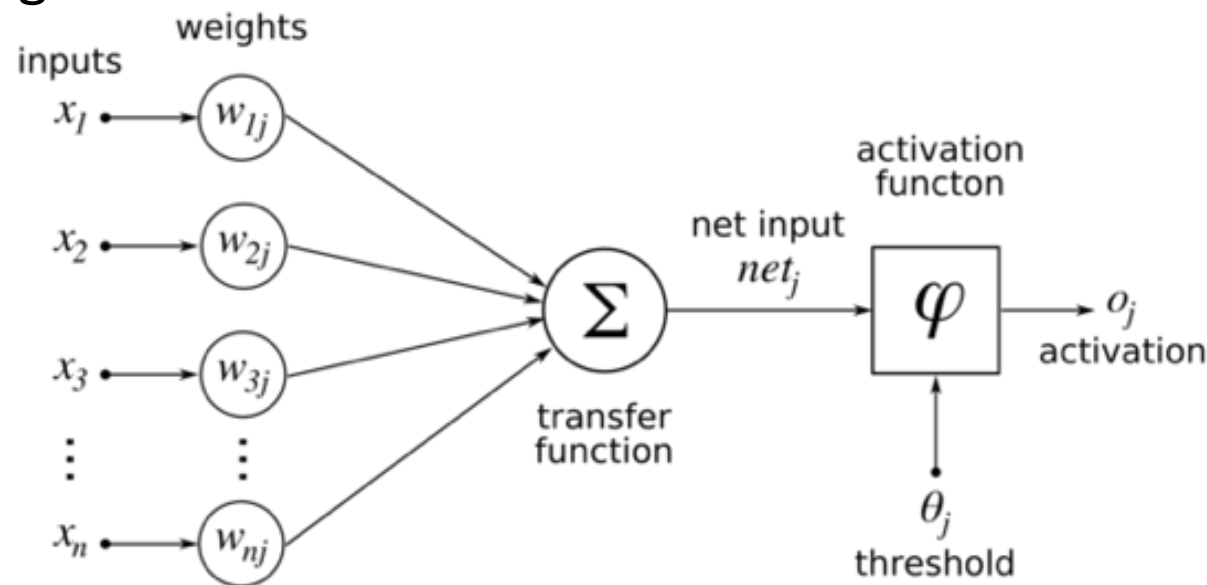
Cada neurónio na rede aplica aos dados duas funções:

## Função de transferência:

- Faz a soma dos produtos das entradas ( $x_i$ ) pelos pesos correspondentes ( $w_i$ )
- adiciona um viés

## Função de ativação $\varphi(x)$

- para obter a sua saída que será a entrada para os neurónios do nível seguinte aos quais está ligado



$$y = \varphi \left( \sum_{i=1}^n w_i x_i \right)$$

$$y = \varphi(w^T x)$$

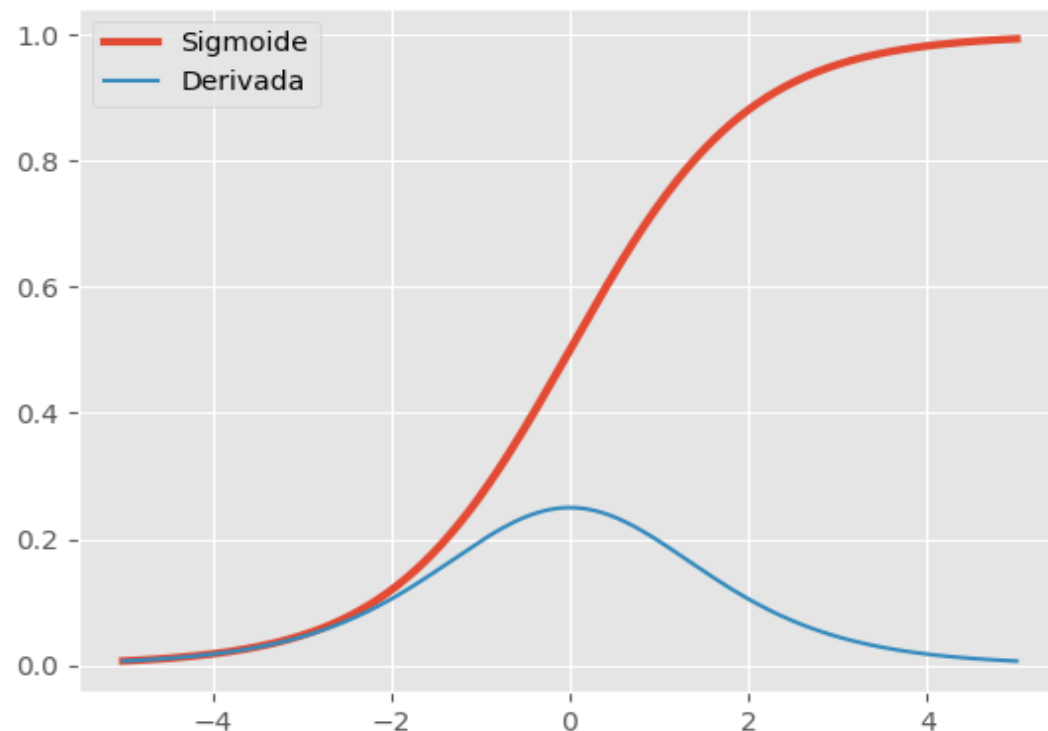
# Função de Ativação

Decide se um neurónio deve ser ativado ou não, ou seja, se a informação que o neurónio está a receber é relevante para a previsão ou se deve ser ignorada

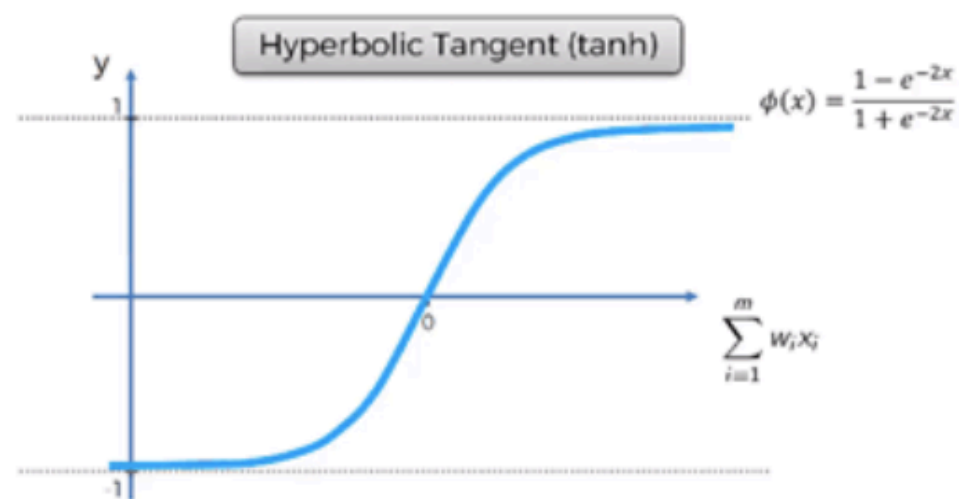
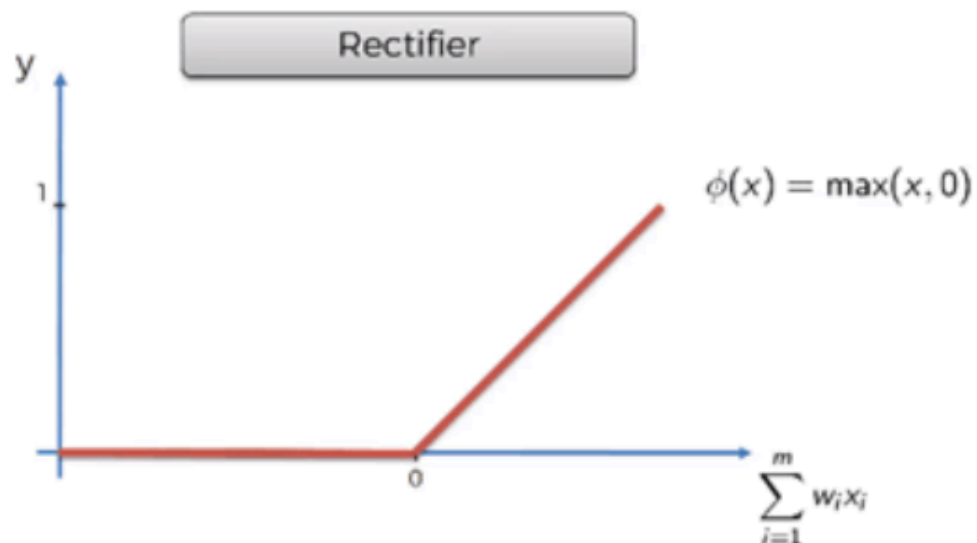
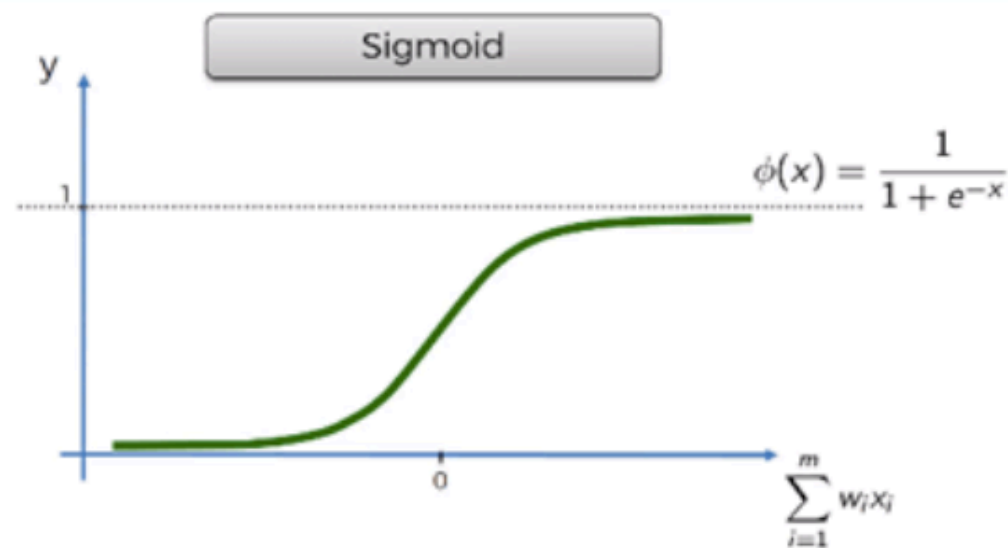
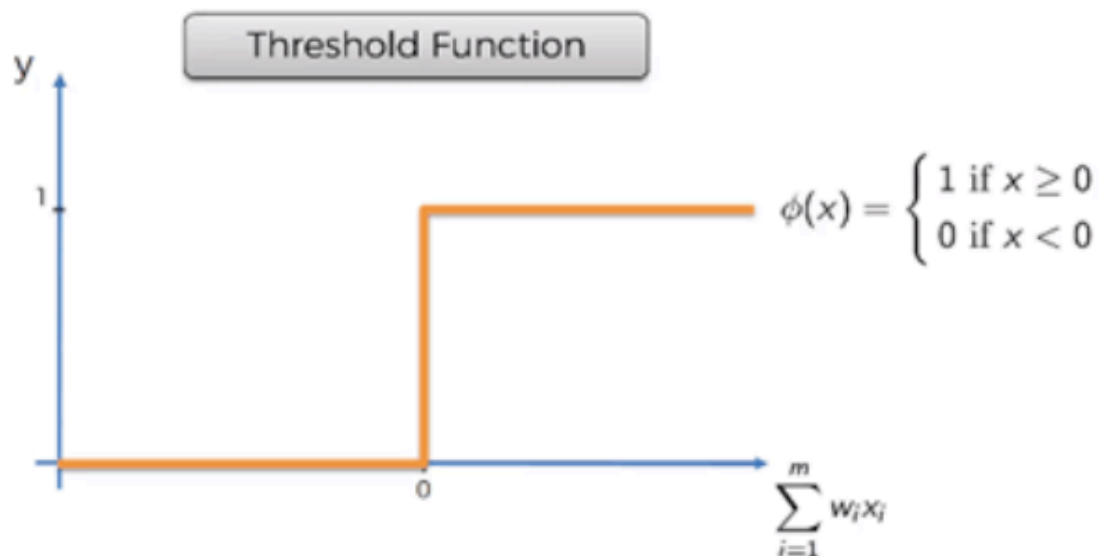
## Função sigmoide

$$\varphi = \frac{1}{1 + e^x}$$

A f. de ativação permite que a rede resolva problemas não lineares

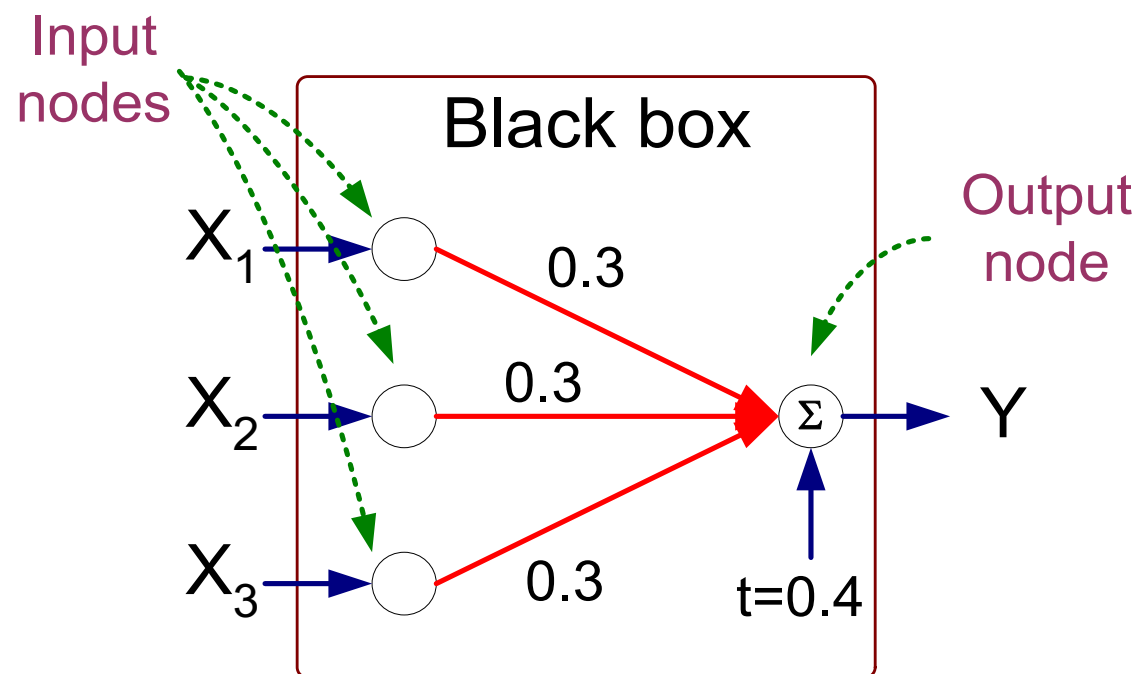


# Funções de ativação mais populares



# Rede Neuronal – Caixa Negra

$X_1$	$X_2$	$X_3$	$Y$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Uma rede neural é um processador que calcula uma variável de saída  $y$  em função de variáveis de entrada  $x_1, x_2, \dots, x_n$

- As variáveis de entrada correspondem aos **atributos previsores** da amostra de dados
- A variável de saída pode ser **discreta (classificação)** ou **contínua (regressão)**

# Número de neurónios na camada de Saída

Depende da variável objetivo

- **Regressão:** um único neurónio gera as previsões de números contínuos
- **Classificação binária:** Um único neurónio gera 1/0 indicando a classe
- **Classificação multi-classe:** número de neurónios igual ao número de classes, cada um representando a saída de uma classe



# Caraterização das Redes Neurais

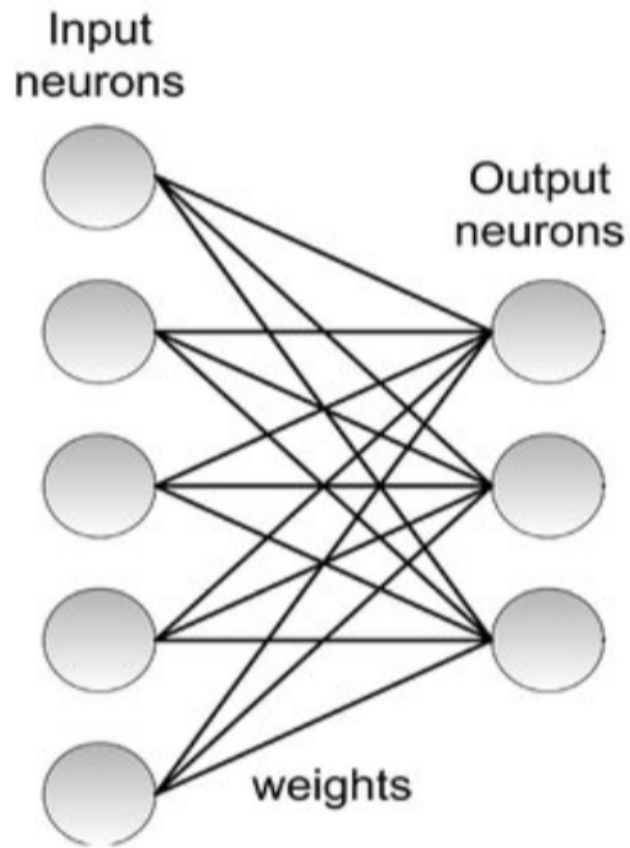
**Topologia** – define o tipo de ligações entre os nós da rede

- **Redes feedforward** – redes só com ligações para a frente
- **Redes recurrent** – redes com conexões de realimentação

**Arquitetura** – define o número de níveis intermédios

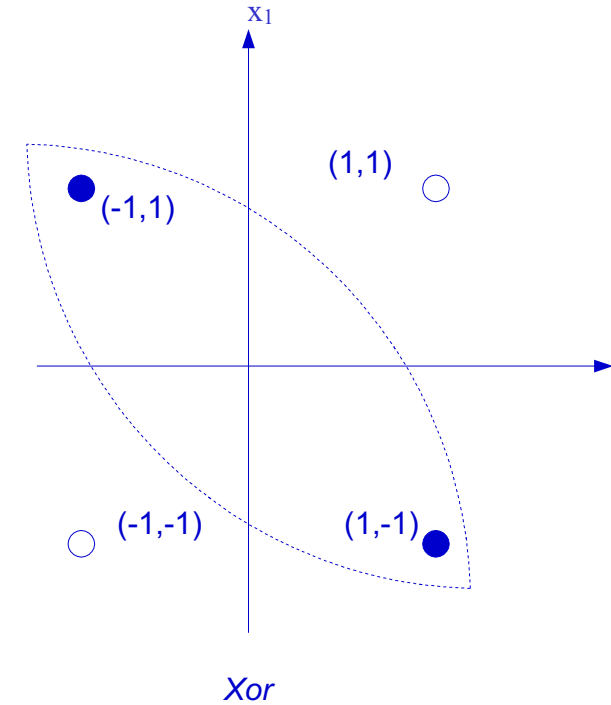
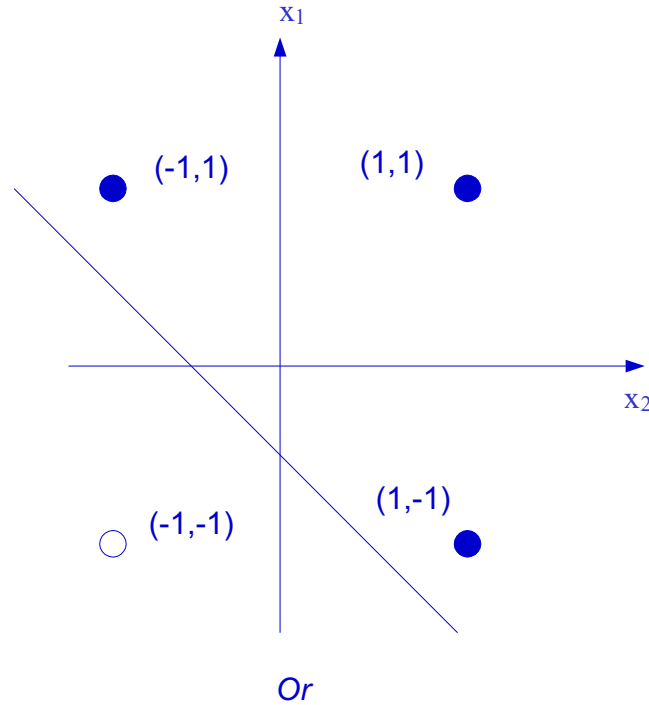
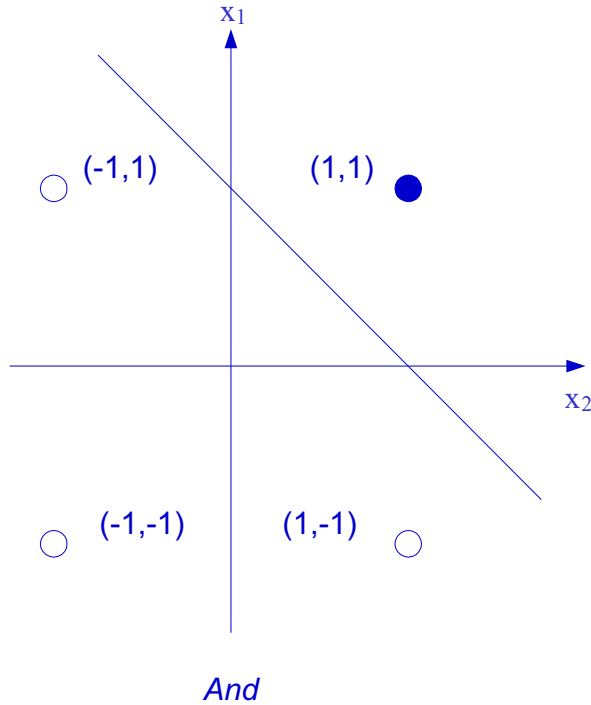
- **Redes Perceptrão** – não existe qualquer nível intermédio, apenas o nível de entrada e de saída
- **Redes Perceptrão Multi-nível** – apresentam um ou mais níveis intermédios

# Rede Perceptrão: um só Nível



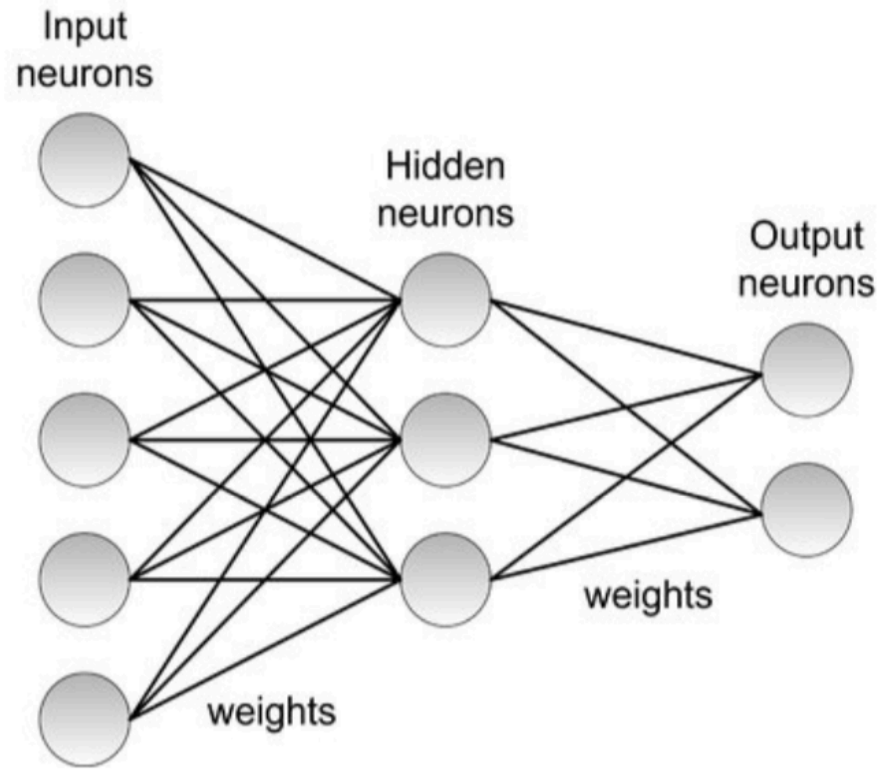
Permite classificar apenas padrões linearmente separáveis (padrões em lados opostos de um hiperplano)

# Limitações Rede Perceptrão



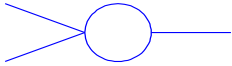
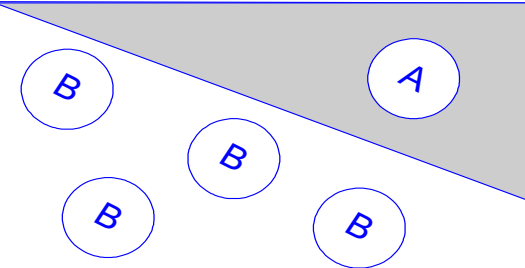
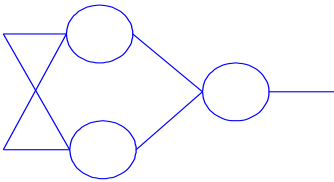
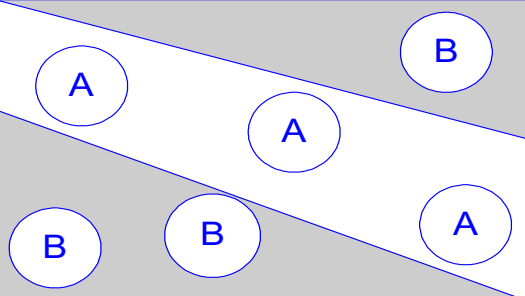
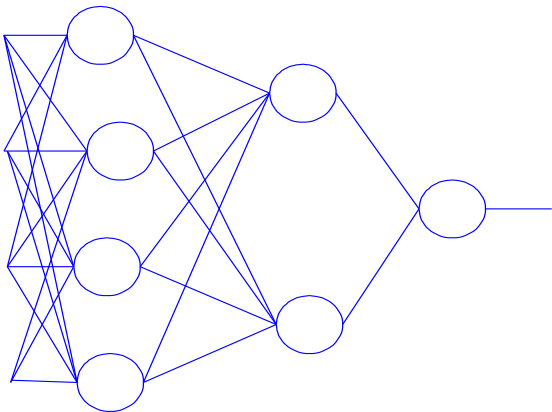
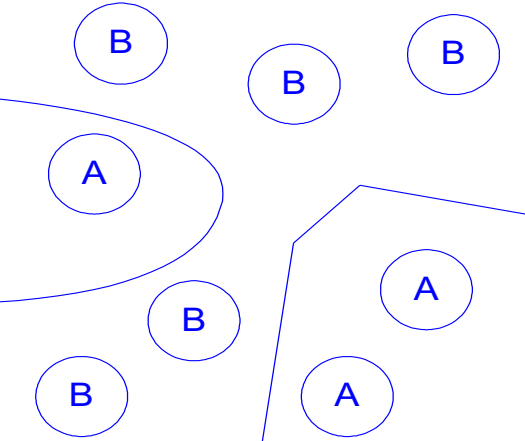
Rede perceptrão não consegue representar a função XOR

# Rede Percepção Multi-níveis (MLP)



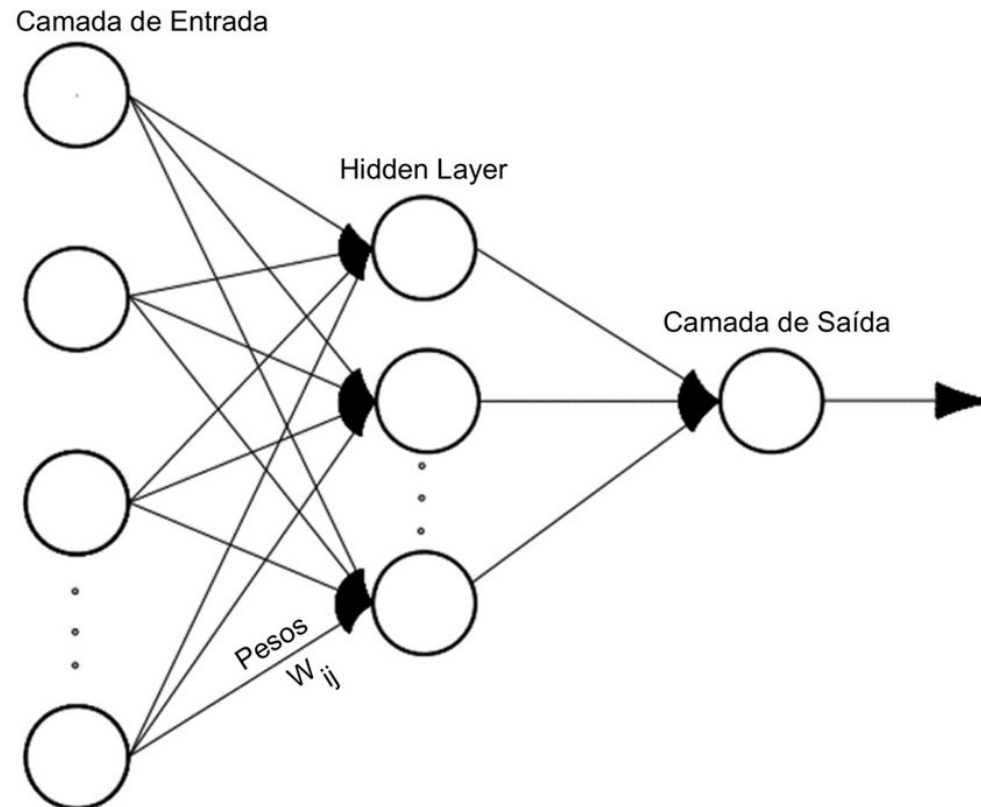
- A rede MLP é composta por uma camada de entrada, uma ou mais camadas intermédias (ou ocultas) e uma camada final — camada de saída
- Cada camada, exceto a camada de saída está totalmente conectada à próxima camada

# Região de Decisão versus Estrutura da Rede

Estrutura		Tipo de Região de Decisão	
Uma Camada		Semi-plano	
Duas camadas		Regiões convexas abertas ou fechadas	
Três Camadas		Qualquer Tipo de Região	

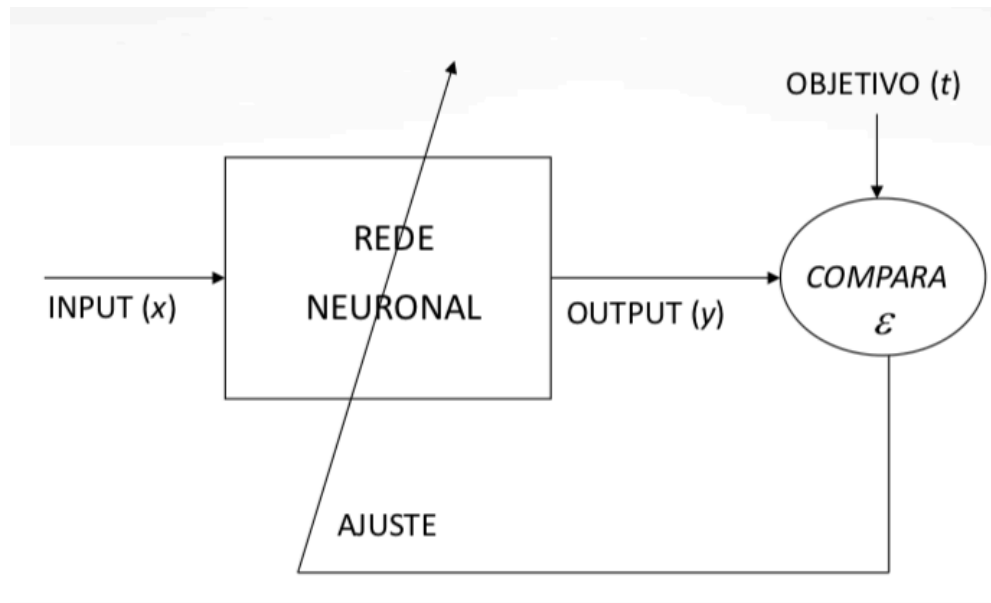
# Rede Neuronal

- É constituída por uma série de nós (ou **neurónios**)
- arranjados em **níveis/camadas**
- interligados (através de conexões com **pesos** numéricos)



# Aprendizagem

O processo de aprendizagem da rede consiste no **ajustamento dos pesos** das ligações entre os neurónios durante o processo de treino da rede

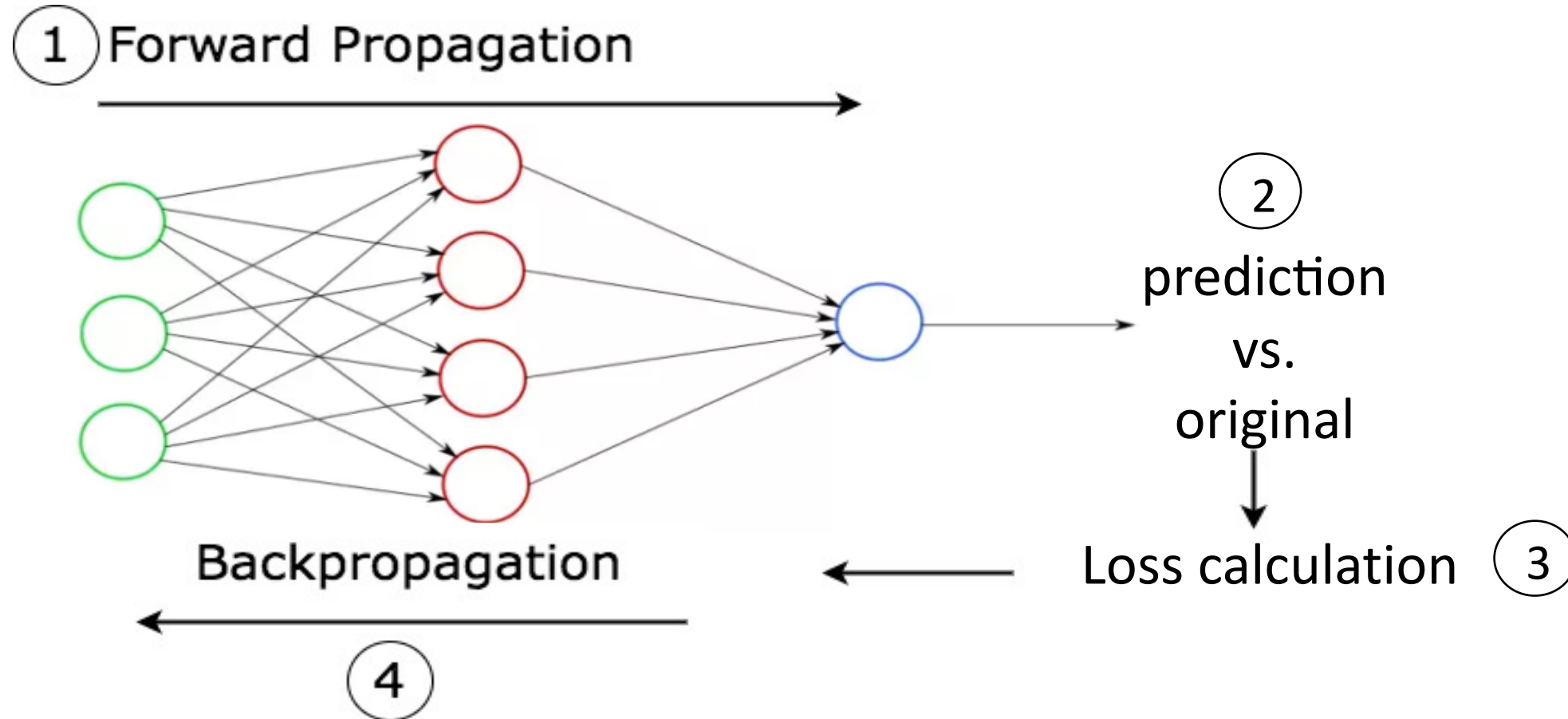


Usa-se:

- um **conjunto de treino** para evitar ajuste exagerado ou deficiente dos pesos ao problema
- um **conjunto de validação** para avaliar o desempenho da rede durante o treino

# Processo de Aprendizagem da Rede Neuronal

A Rede aprende por múltiplas iterações de propagação para frente e para trás



Este ciclo é repetido para cada registo dos dados



# Algoritmo de Aprendizagem Retroativa

## *Backpropagation Algorithm*

Inicializa aleatoriamente os pesos da rede

### **Repete**

1. calcula o output  $y(x)$  para os inputs  $x_1, x_2, \dots, x_n$  e pesos atuais da rede
2. determina o erro da previsão
3. propaga o erro para trás na rede através da adaptação dos pesos da rede

**Até** (erro atingir um mínimo ou um limite máximo de iterações)

# Atualização dos pesos da rede

A atualização dos pesos da rede faz-se através da minimização do erro dado pela função:

- **soma do quadrado dos erros** para a regressão
- **taxa de acerto** para classificação

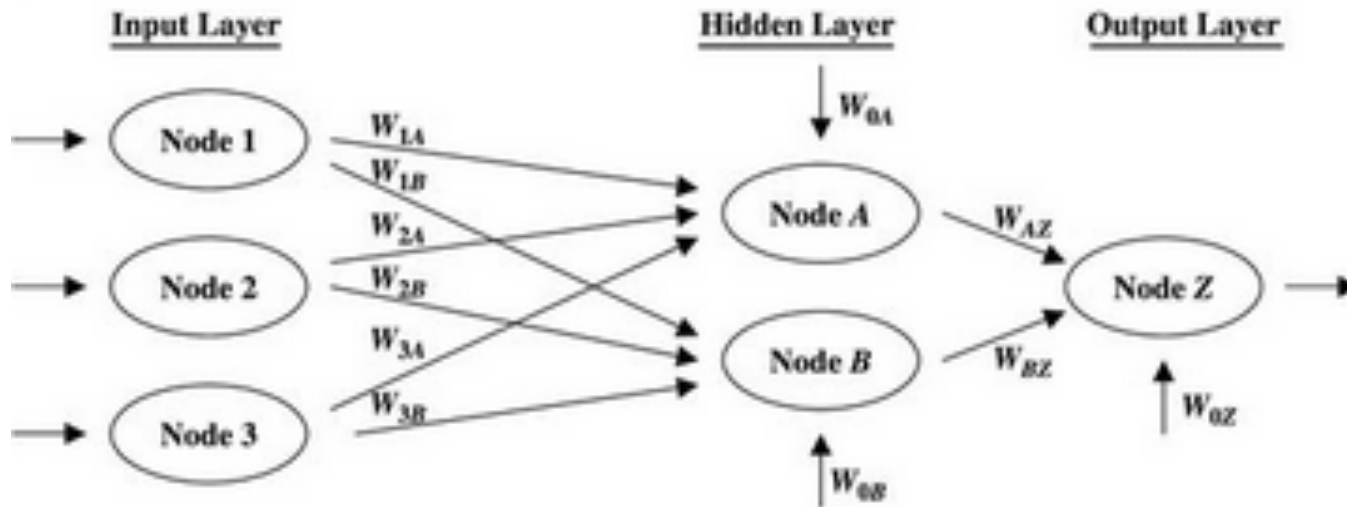
O treino da rede pode ser feito usando:

- **Abordagem batch** (*Batch Gradient Descent*)  
a atualização dos pesos é feita após a apresentação à rede de todos os casos do conj<sup>to</sup> de treino
- **Abordagem semi-batch** (*Mini Batch Gradient Descent*)  
a atualização dos pesos é feita depois da rede aprender alguns casos do conj<sup>to</sup> de treino: 5, 10, 50 linhas, etc.
- **Abordagem online** (*Stochastic Gradient Descent*)  
a atualização dos pesos é feita após a apresentação de cada caso do conj<sup>to</sup> de treino

# Exemplo funcionamento da rede neuronal

### Data Inputs and Weights:

$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	



Input Attributes:  $x_1, x_2, x_3$

Predicted Value: 0.8750

$$\begin{aligned}\text{net}_A &= \sum_i W_{iA}x_{iA} = W_{0A}(1) + W_{1A}x_{1A} + W_{2A}x_{2A} + W_{3A}x_{3A} \\ &= 0.5 + 0.6(0.4) + 0.8(0.2) + 0.6(0.7) = 1.32\end{aligned}$$

$$f(\text{net}_A) = \frac{1}{1 + e^{1.32}} = 0.7892$$

$$\begin{aligned}\text{net}_B &= \sum_i W_{iB}x_{iB} = W_{0B}(1) + W_{1B}x_{1B} + W_{2B}x_{2B} + W_{3B}x_{3B} \\ &= 0.7 + 0.9(0.4) + 0.8(0.2) + 0.4(0.7) = 1.5\end{aligned}$$

$$f(\text{net}_B) = \frac{1}{1 + e^{-1.5}} = 0.8176$$

$$\begin{aligned}\text{net}_Z &= \sum_i W_{iZ}x_{iZ} = W_{0Z}(1) + W_{AZ}x_{AZ} + W_{BZ}x_{BZ} \\ &= 0.5 + 0.9(0.7892) + 0.9(0.8176) = 1.9461\end{aligned}$$

$$f(\text{net}_Z) = \frac{1}{1 + e^{-1.9461}} = 0.8750$$

# Arquitetura típica da Rede MLP para Regressão

- **Input neurons:** um por cada atributo de previsão
- **Hidden layers:** dependente do problema, tipicamente entre 1 e 5
- **neurons per hidden layer:** dependente do problema, tipicamente entre 10 e 100
- **Output neurons:** 1 por cada variável a prever
- **Hidden activation function:** ReLU
- **Output activation:** ReLU (se saída positiva) /softplus  
logística/tanh (se saída limitada a um intervalo de valores)
- **Loss function:** MSE or MAE

# Arquitetura típica da Rede MLP para Classificação

- **Input neurons:** um por cada atributo de previsão
- **Hidden layers:** dependente do problema, tipicamente entre 1 e 5
- **neurons per hidden layer:** dependente do problema, tipicamente entre 10 e 100
- **Output neurons:** Classificação binária — 1 neurónio  
Classificação multi-classe — 1 neurónio por label
- **Hidden activation function:** ReLU
- **Output activation:** Classificação binária — Logistic  
Classificação multi-classe — Softmax
- **Loss function:** Cross entropy

# Implementação de MLPs com Keras

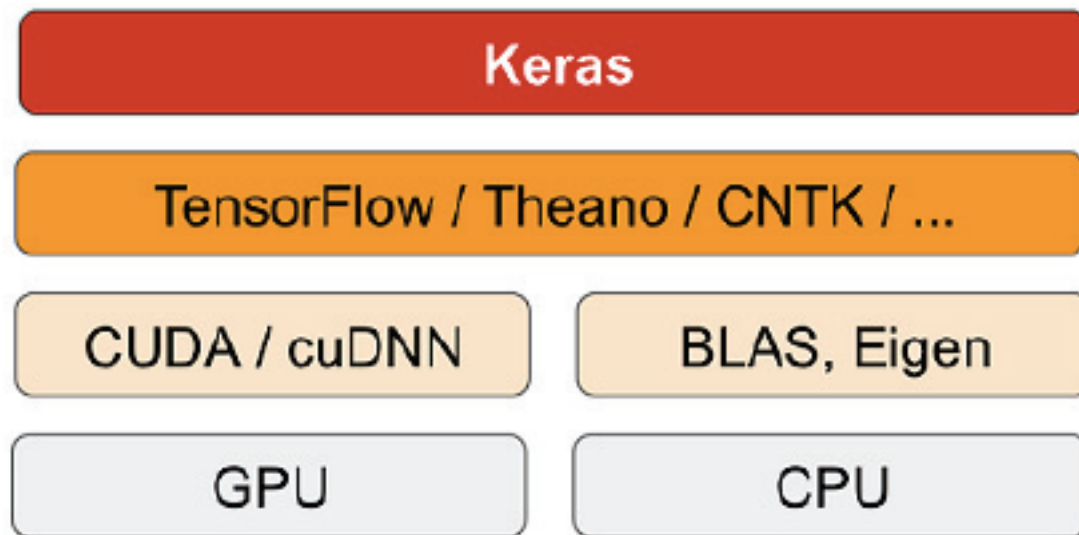
- Rede – é o foco do Keras
- Uma rede em Keras é uma sequência de camadas – **Sequencial**
- **Sequencial**: é uma pilha linear de camadas

Passos para a construção de redes em Keras:

1. **Definir a rede**: Criar uma rede sequencial e adicionar camadas configuradas
2. **Compilar a rede**: Especificar a função de perda (loss function), otimizadores e chamar o método **compile()**
3. **Ajustar a rede**: Treinar a rede numa amostra de dados chamando a função **fit()** no modelo
4. **Fazer previsões**: Usar a rede para gerar previsões sobre novos dados chamando as funções **evaluate()** ou **predict()**

# Implementação de MLPs com Keras

# Keras – API



**Keras** é uma **API Deep Learning** de alto nível, em Python que pode ser executada em qualquer um destes três frameworks:

- TensorFlow (da Google)
- CNTK (da Microsoft)
- Theano (do Montreal Institute for Learning Algorithms, Université Montréal, Canadá)



# Preparação dos Dados

## Dados numéricos

- Variáveis categóricas:
  - 2 classes representadas por 0/1
  - $k$  classes são representadas por  $k$  variáveis binárias (*dummy variables*)

## Dados normalizados

- Normalização minmax:

$$y' = \frac{y - \min_y}{\max_y - \min_y}$$

- Normalização Zscore:

$$y' = \frac{y - \text{média}_y}{\text{desvio.padrão}_y}$$

# Conjuntos de Treino, Validação, Teste

As previsões são desenvolvidas no **conjunto de treino**

Obter previsões certas envolve sempre ajustar a configuração da rede:

- escolher o número de camadas e o tamanho das camadas, nº de neurónios - **hiperparâmetros**
- escolher os pesos da rede - **parâmetros**

Esta afinação é repetida muitas vezes e avaliada com o **conjunto de validação**

O desempenho final da rede deve ser avaliado num conjunto de dados completamente novo - **conjunto de teste**

# Keras workflow

1. Definir os dados de treino, teste e validação
2. Definir as camadas da rede que mapeia as entradas para as saídas
3. Configurar o processo de aprendizagem da rede escolhendo:
  - i. **função de ativação**
  - ii. **função de perda** (função objetivo) — a quantidade que será minimizada durante o treino. Representa uma medida de sucesso para a tarefa a executar
  - iii. **otimizador** — determina como a rede será atualizada com base na função de perda. Implementa uma variante específica do gradiente estocástico descendente
4. Processar os dados de treino usando a f. `fit()` com a rede treinada

# Implementação MLP para classificação com Keras

```
from keras.models import Sequential
from keras.layers import Dense

nnet = Sequential()
nnet.add(Dense(units=15,
                input_dim=X_train.shape[1],
                kernel_initializer='uniform',
                activation='relu'))

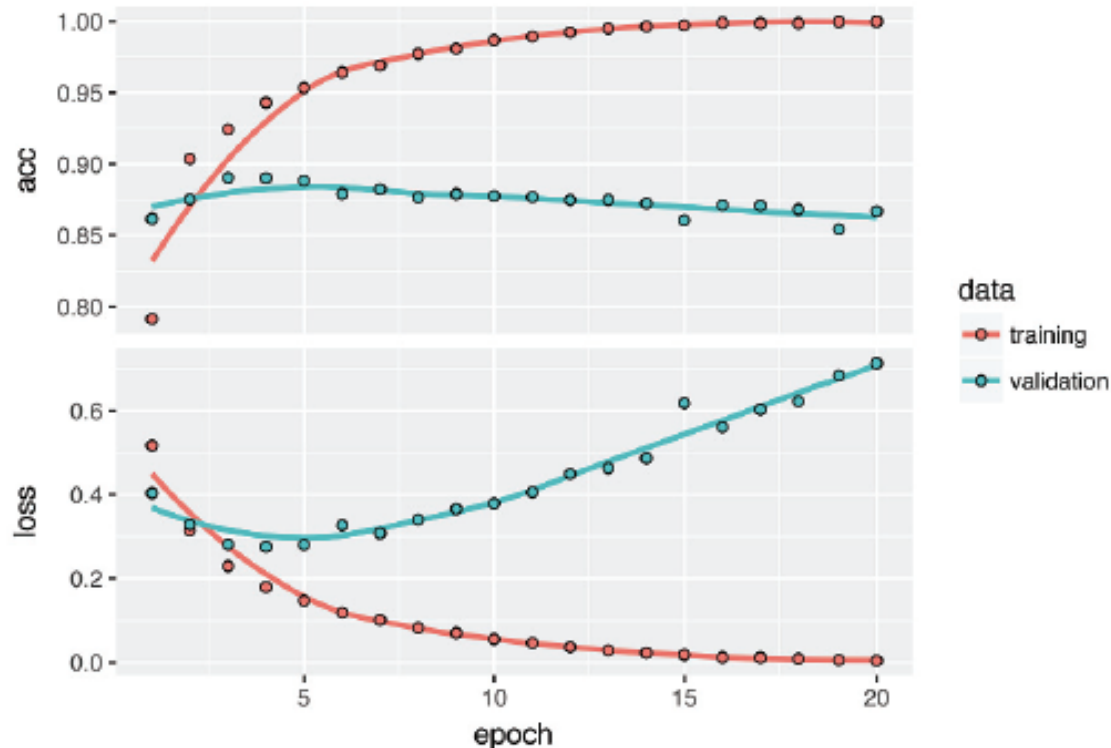
nnet.add(Dense(units=15,
                kernel_initializer='uniform',
                activation='relu'))

nnet.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))

#Compiling the nnet
nnet.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# fitting the Neural Network
history = nnet.fit(X_train, y_train, validation_split=0.1, batch_size=200, epochs=50)
```

# Gráficos com as métricas de treino e validação



- **Loss de treino** diminui e **accuracy treino** aumenta em cada época — é o esperado quando se faz uma **otimização gradiente descente**
- Mas, a **accuracy de validação** baixa a partir da 4ª época de treino e o **loss de validação** aumenta — **overfitting**
- Existem várias técnicas para evitar o **overfitting**

# Evitar o *overfitting*

- **Obter mais dados de treino**
- **Reduzir a capacidade da rede:** o número de camadas e o número de neurónios por camada
  - Começar com relativamente poucas camadas e neurónios e aumentar iterativamente o número de camadas /neurónios
- **Adicionar regularização de pesos**
  - É feito adicionando à função de perda da rede um custo associado a ter grandes pesos
- **Adicionar dropout (desistência)**
  - descartar aleatoriamente (definir como zero) recursos de saída das camadas internas durante o treino

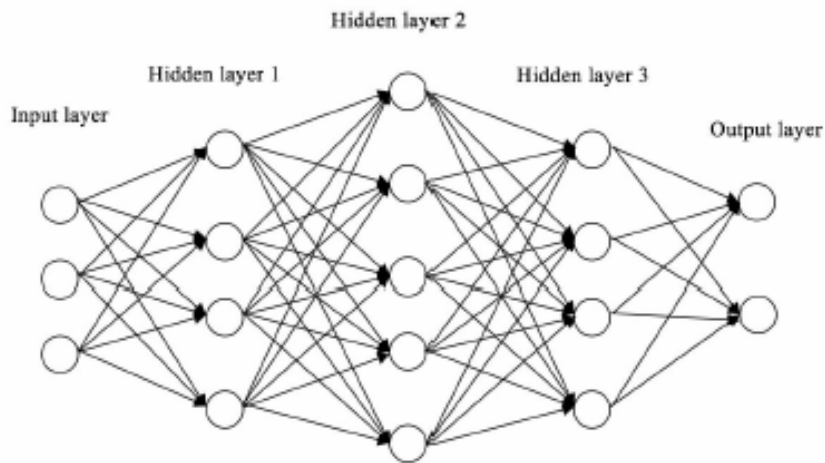
# Prevenir overfitting: Regularização dos pesos

Overfitting pode ser reduzido colocando restrições na complexidade da rede - forçando valores pequenos para os pesos da rede - torna a distribuição de valores de pesos mais regular - **regularização de peso**

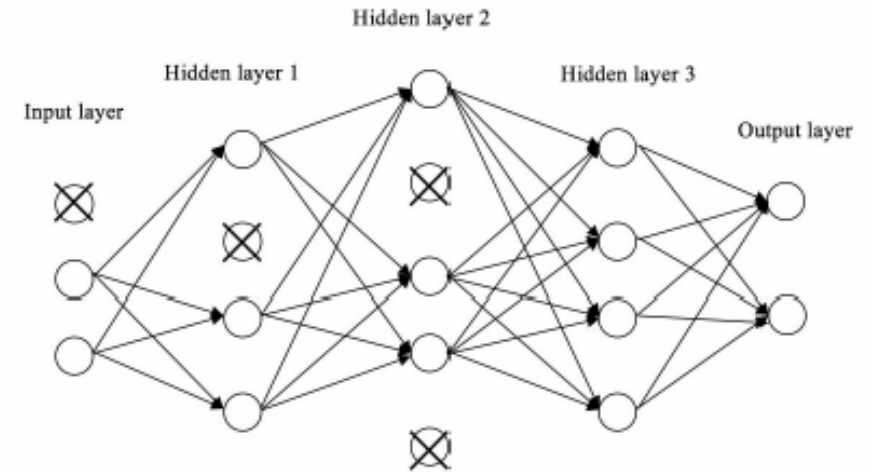
- **Regularização dos pesos** é feita adicionando à função de perda da rede um custo associado a ter pesos grandes
- Existem dois tipos de regularização:
- **Regularização L1** —O custo adicionado é proporcional ao valor absoluto dos coeficientes de peso (a norma L1 dos pesos)
- **Regularização L2** —O custo adicionado é proporcional ao quadrado do valor dos coeficientes de peso (a norma L2 dos pesos)

# Prevenir overfitting: Dropout

**Dropout** consiste em descartar aleatoriamente (definir como zero) uma série de recursos de saída das camadas internas durante o treino



Standard NN



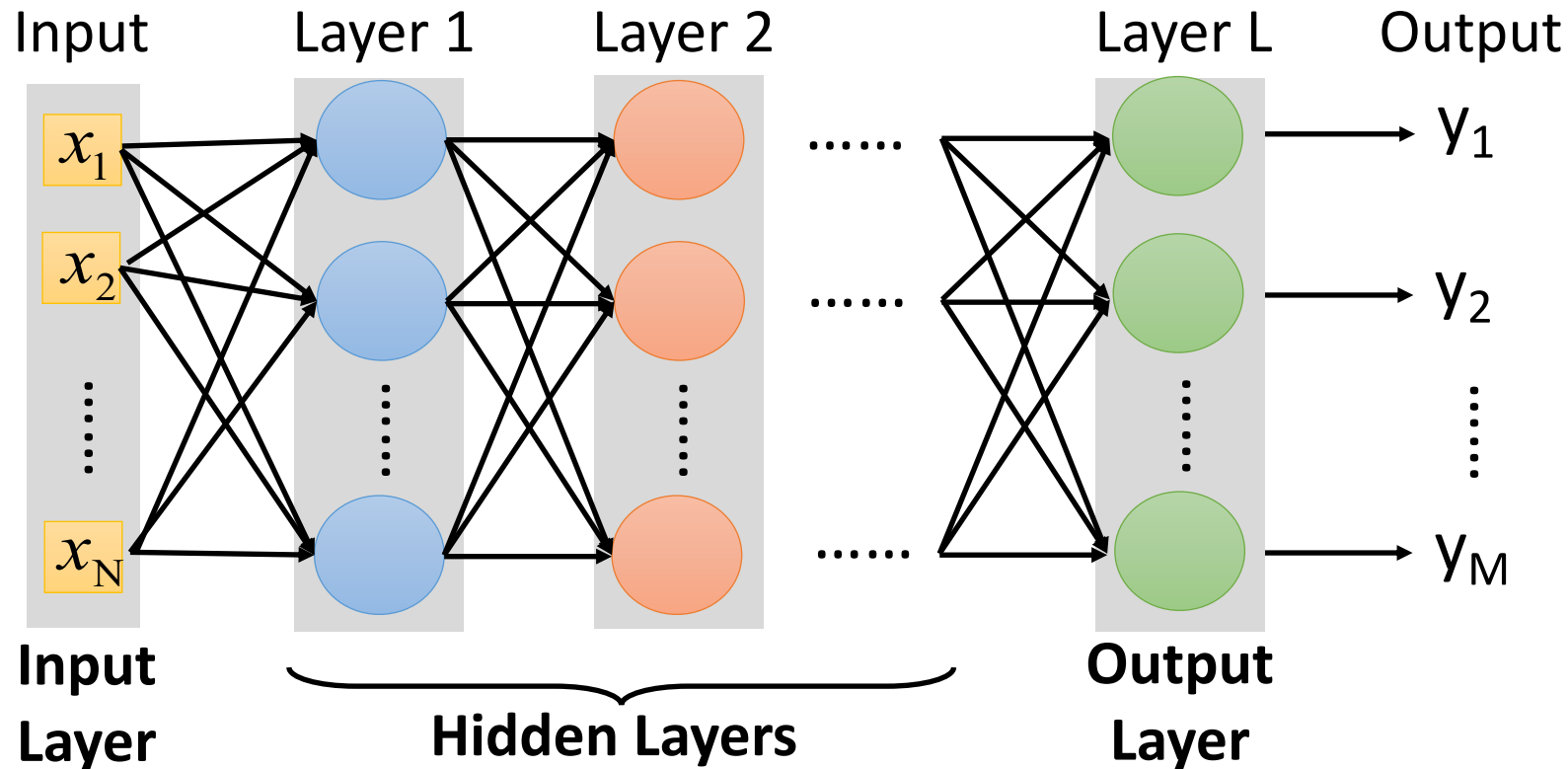
NN com dropout

- **Taxa de dropout:** é a fração das características que são zeradas; geralmente toma valores entre 0,2 e 0,5
- **No teste:** nenhuma unidade é descartada



# Deep Learning

# Rede Neuronal vs. Rede Deep Learning



As redes profundas ou redes *deep learning* distinguem-se das RNs por terem muitas camadas ocultas

Deep significa muitos “níveis escondidos”

# Tipos de redes Deep Learning

## Deep Learning Supervisionado

- Artificial Neural Networks (ANN)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- Long Short Term Memory Networks (LSTM)

## Deep Learning não Supervisionado

- Self Organising Maps (SOM)
- Restricted Boltzmann machines (RBM)
- Autoencoders
- Deep Belief Networks (DBN)

# Aplicações práticas CNN e RNN/LSTM

Aplicações de redes CNNs - visão computacional:

- Reconhecimento facial, reconhecimento de objetos em imagens

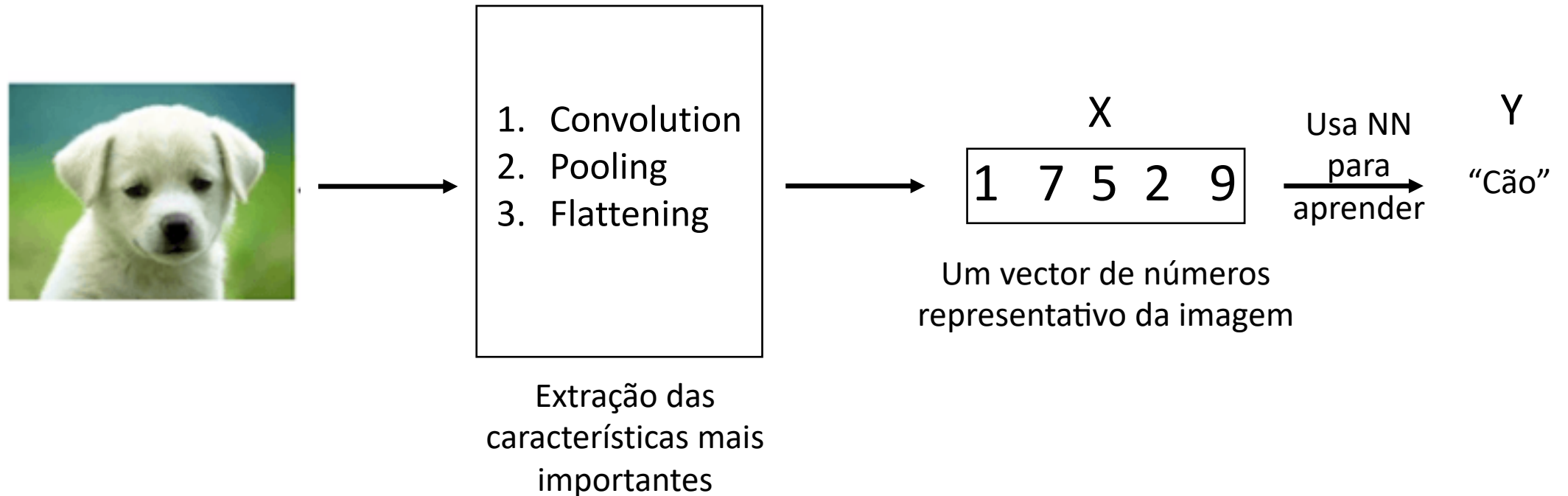
Aplicações de redes RNN/LSTM:

- Análises séries temporais
- Processamento automático de texto:
  - Tradução de idiomas (inglês para francês, francês para chinês, etc.)
  - Chatbots
  - Legenda de imagens (resumo do que está presente numa imagem)
  - Legendas de filmes
  - Reconhecimento de fala: conversão de fala em texto
  - Voz de computador: conversão de texto em fala

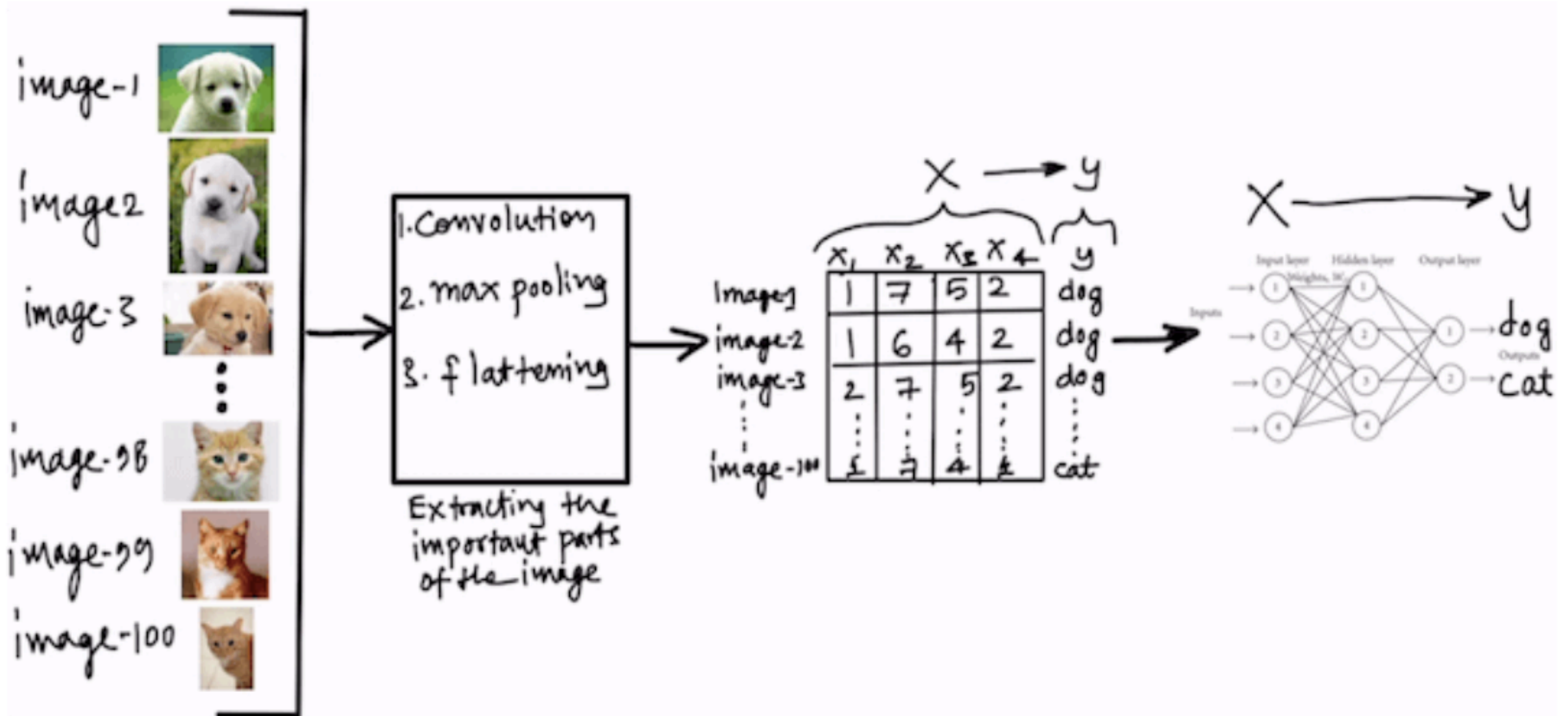
# Convolutional Neural Network

# Convolutional Neural Network (CNN)

- Combina várias etapas para derivar as características mais importantes de uma imagem
- Em seguida, converte a imagem inteira numa única linha de características que são depois apreendidas por uma rede neuronal totalmente conectada

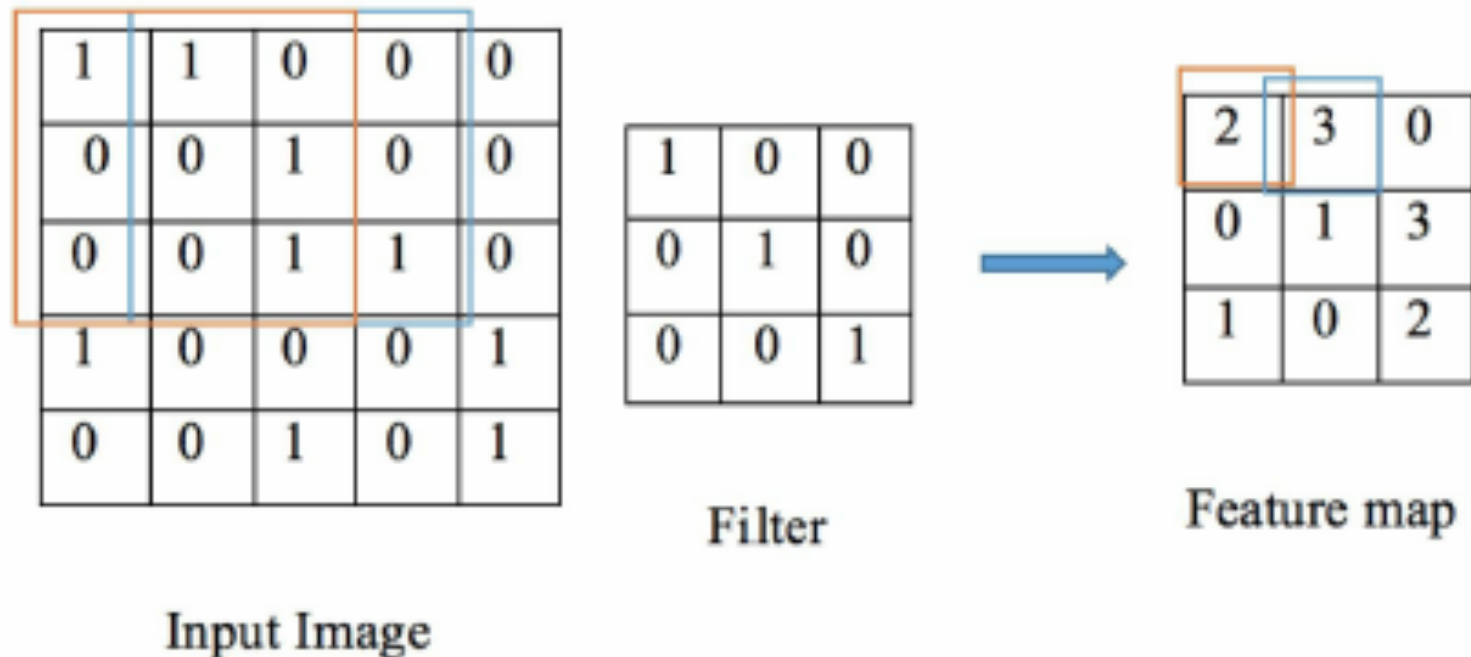


# Fluxo geral de uma rede neural convolucional (CNN)



# Passo 1: Convolution

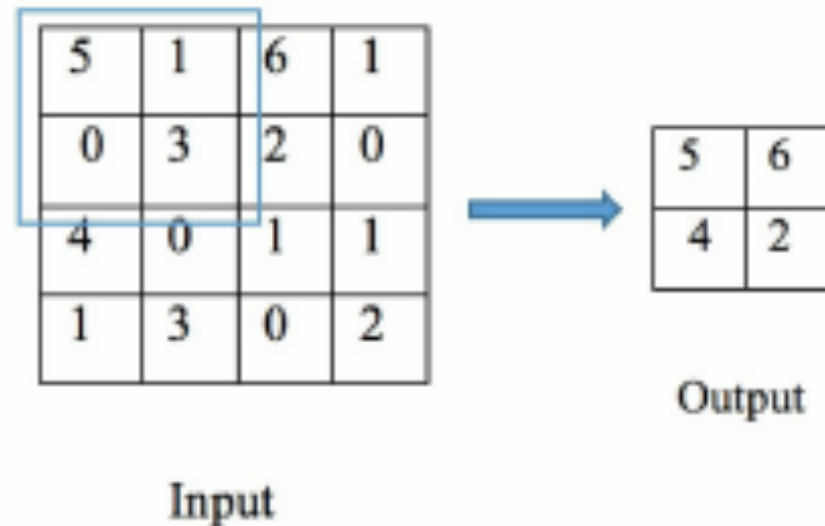
- Permite converter dados de imagem não estruturados em dados estruturados
- As informações digitais da imagem são resumidas extraíndo os pixéis mais importantes da imagem através de **filtros** também conhecidos como **kernels**





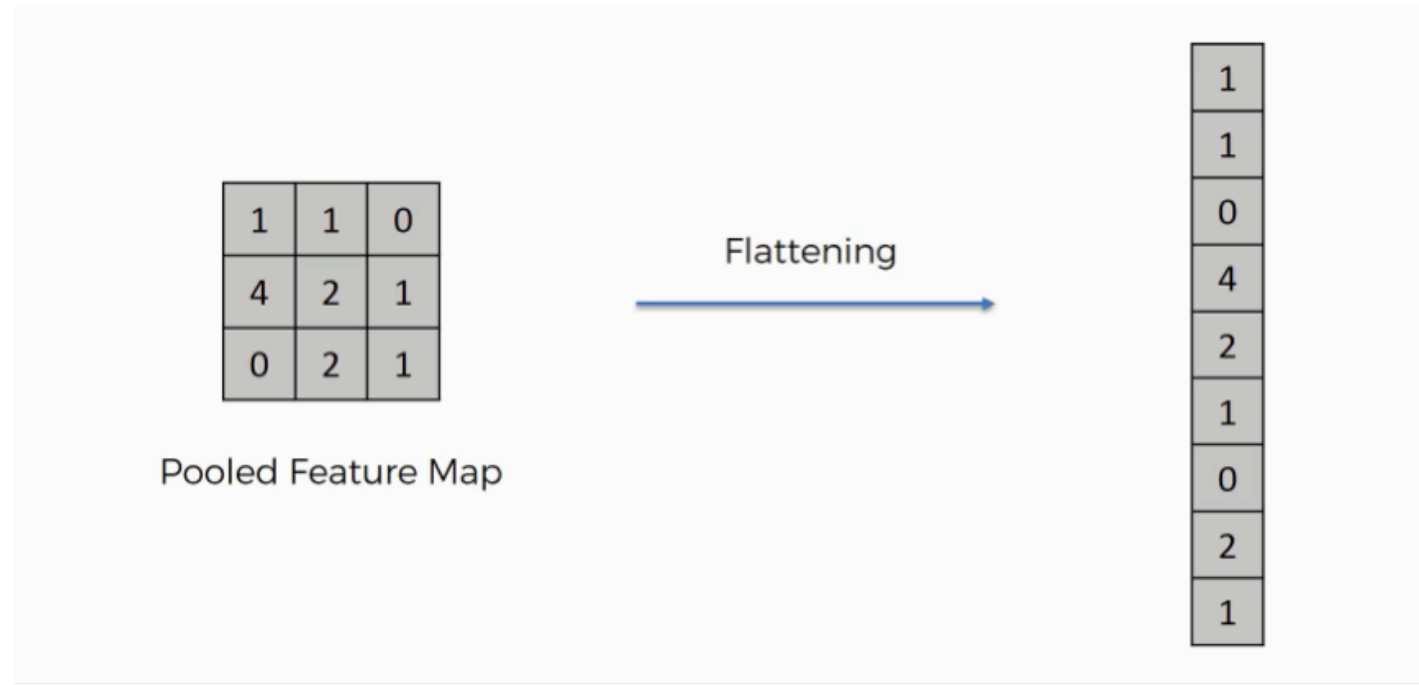
# Passo 2: Pooling

- Agrega estatísticas por sub-regiões para gerar matrizes menores
- Métodos típicos de pooling incluem agrupamento máximo e agrupamento médio, que usam os valores máximos e os valores médios de todas as sub-regiões não sobrepostas
- Exemplo, de um filtro de agrupamento máximo  $2 \times 2$  num mapa de recursos  $4 \times 4$ :



# Passo 3: Flattening

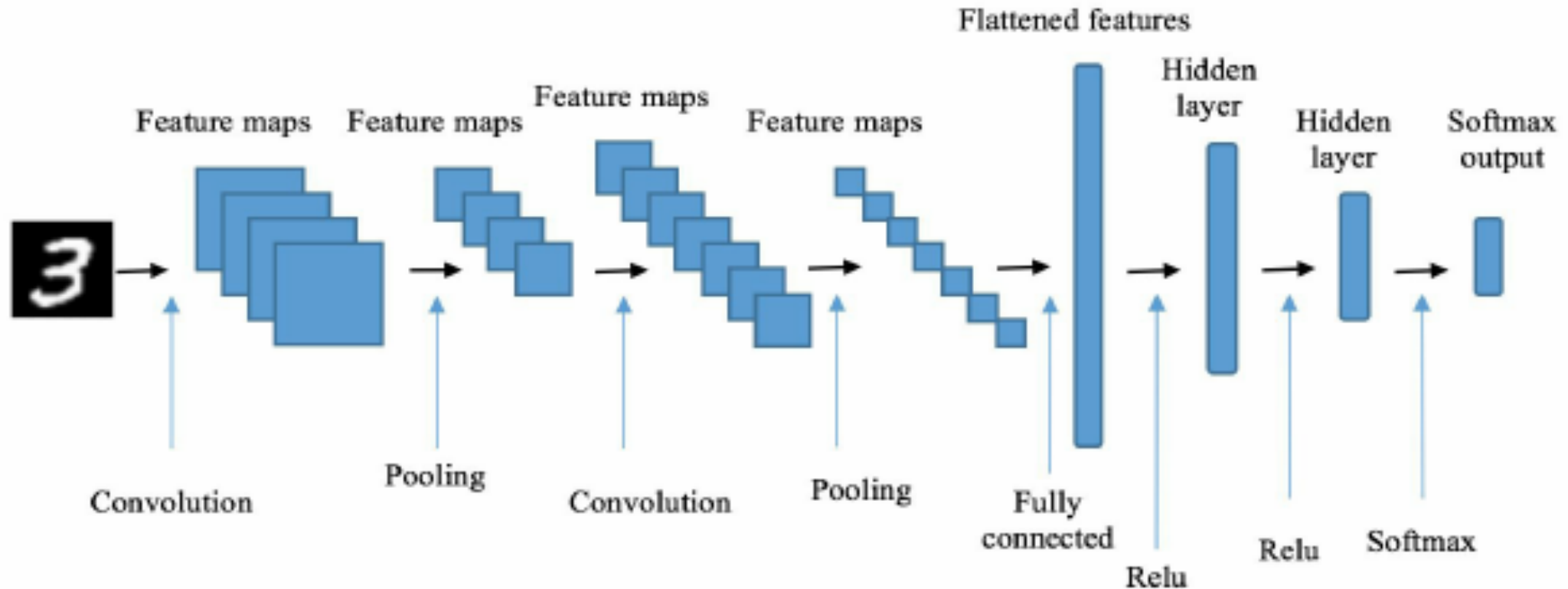
- Converte a matriz resultante do passo anterior – pooling num vetor para ser passado à rede para fazer a aprendizagem



# Convolutional Neural Network (CNN)

Uma rede CNN é constituída:

- por uma sequência de camadas convolucionais e camadas de pooling/agrupamento
- seguida de uma rede totalmente conectada para gerar as probabilidades de cada classe



# Redes CNN pré-treinadas

Uma rede pré-treinada é uma rede que foi previamente treinada numa tarefa de classificação de imagens em larga escala

Uma rede pré-treinada pode atuar como um modelo genérico do mundo visual e ser usada em tarefas de classificação que podem envolver classes completamente diferentes da tarefa original para as quais foram treinadas - **Vantagem importante da aprendizagem profunda**

- VGG16
- VGG19
- ResNet
- Inception
- Inception-ResNet
- Xception

# Referências

- Chollet, F. (2021) Deep learning with Python. Simon and Schuster, Manning Publications, ISBN 9781617294433
- Géron, A. (2022) Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc, ISBN 9781492032649