

# 12 - Prática: Redes Neurais (II)

Obs: Afim de tornar mais compreensível processo, foi realizado documentação junto ao código, tanto no módulo 6 quanto 7

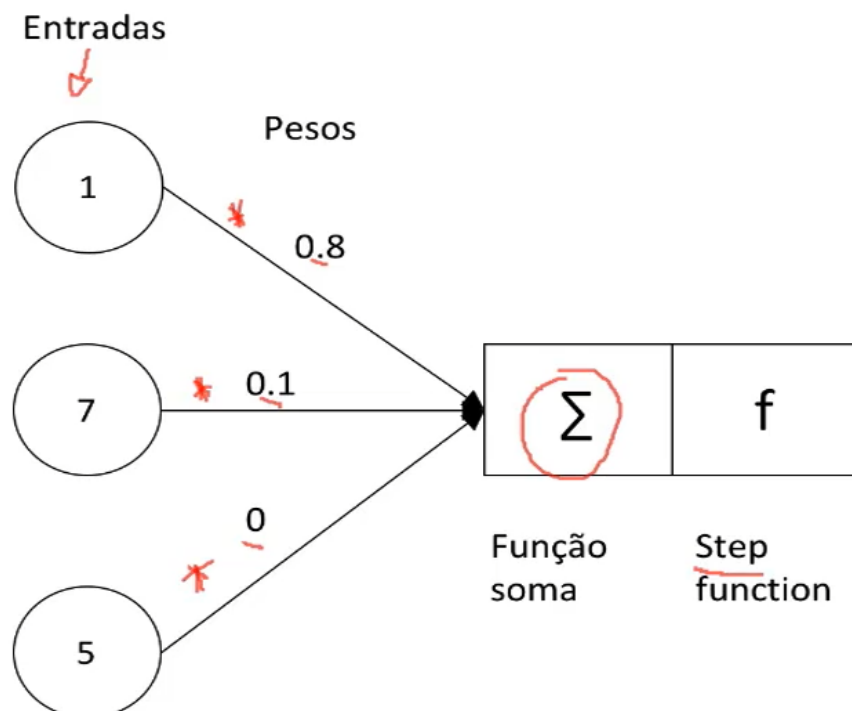
## ▼ Módulo 3

### ▼ Perceptron de uma camada

-

$$soma = \sum_{i=1}^n \underline{x_i} * \underline{w_i}$$

Entrada multiplicada pelos respectivos pesos:



$$soma = \sum_{i=1}^n \underline{x_i} * \underline{w_i}$$

$$soma = (\underline{1 * 0.8}) + (\underline{7 * 0.1}) + (\underline{5 * 0})$$

$$soma = \underline{0.8} + \underline{0.7} + \underline{0}$$

$$soma = \underline{1.5}$$

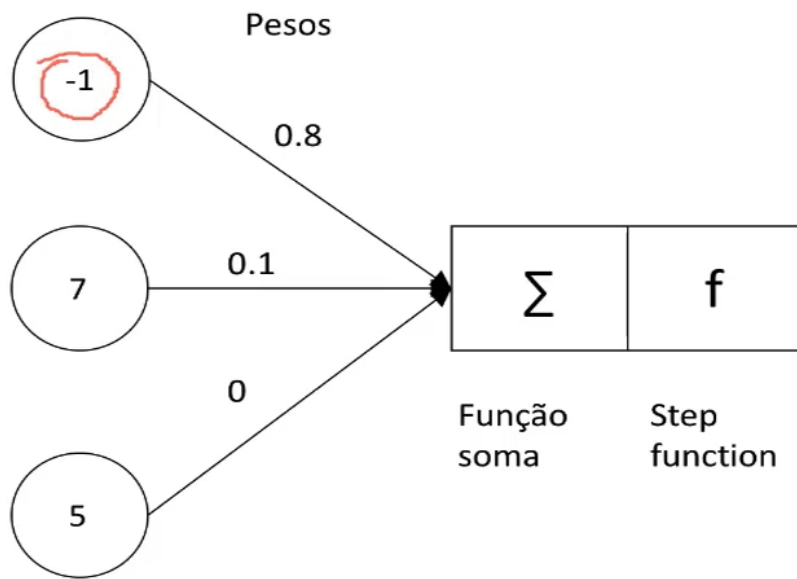
## Step function (função Degrau)

✓ Maior do que zero = 1

Caso contrário = 0

Processo basico de cálculo de uma rede neural. Você pega as entradas, multiplica com os pesos, soma esse valor entre si e caso seja maior que zero terá valor 1 caso contrário valor 0

Entradas



$$soma = \sum_{i=1}^n x_i * w_i$$

$$soma = (-1 * 0.8) + (7 * 0.1) + (5 * 0)$$

$$soma = -0.8 + 0.7 + 0$$

$$soma = -0.1$$

Neste caso acima, o resultado será 0.

$$\begin{array}{r}
 \text{C} \quad \text{P} \\
 0 - 0 = 0 \\
 0 - 0 = 0 \\
 0 - 0 = 0 \\
 1 - 0 = 1
 \end{array}$$

Resposta correta e resposta prevista.

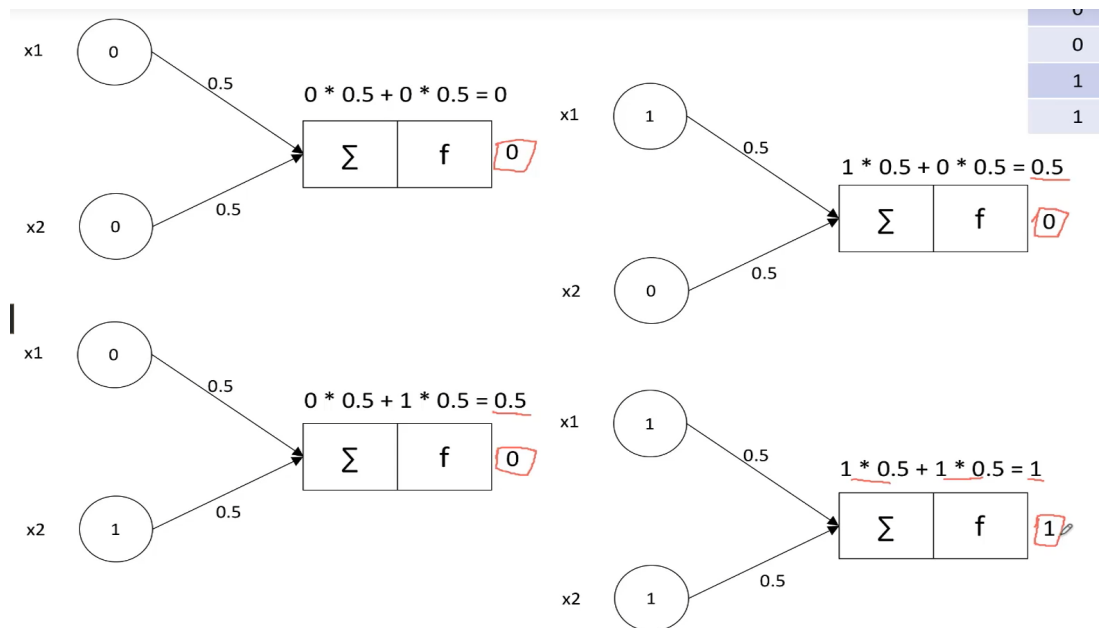
Caso tenhamos diversos neuronios realizando processo de cálculo do resultado dos seus valores entre si, conseguiremos analisar se seus retornos serão corretos relacionando os mesmos com os valores previstos.

No cálculo acima, é possível identificar que 3 dos retornos foram iguais a 0, através disto temos 3 respostas devidas.

Ao validar o ultimo resultado, encontramos valor de 1, sendo o mesmo um erro. É possível dizer então que para estes 4 retornos, existe 75% de acerto.

$$\begin{aligned}
 \text{peso}(n + 1) &= \text{peso}(n) + (\text{taxaAprendizagem} * \text{entrada} * \text{erro}) \\
 \text{peso}(n + 1) &= 0 + (0.1 * 1 * 1) = 0.1
 \end{aligned}$$

Para resolver esse erro, utilizamos o cálculo acima, o mesmo possibilitará a resolução do caso



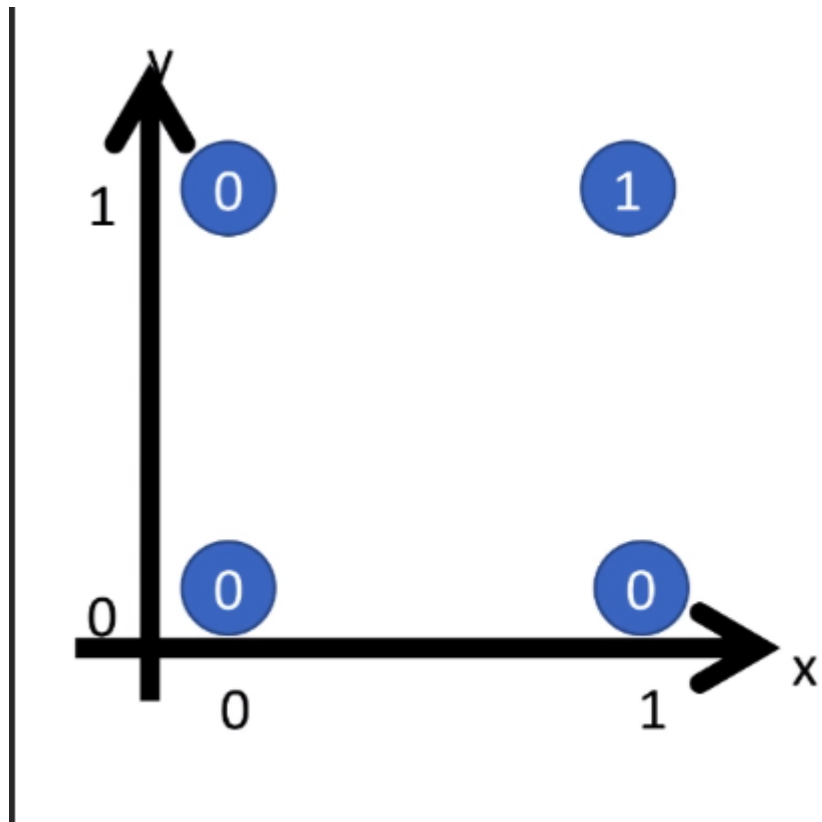
Averiguando novamente cálculo e aplicando o valor de 0,5 que chegamos acima, conseguimos “resolver” a situação da qual nossa resposta foge do devido.

$$\begin{aligned}
 0 - 0 &= 0 \\
 0 - 0 &= 0 \\
 0 - 0 &= 0 \\
 1 - 1 &= 0
 \end{aligned}$$

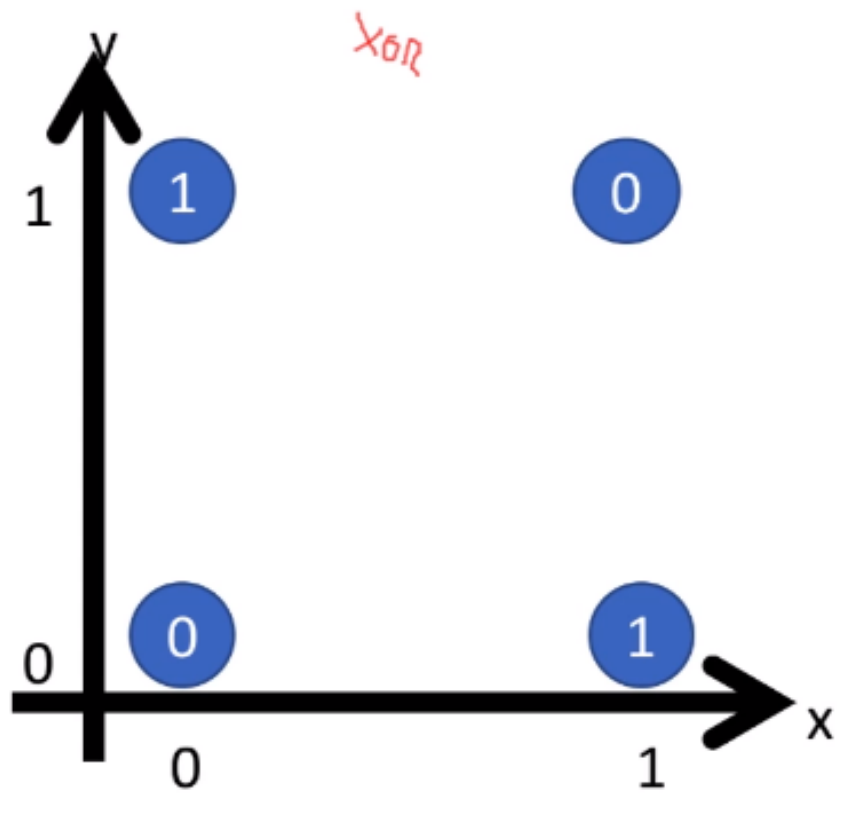
Fazendo novamente cálculo, temos resposta de 100%

**Aprendizagem de uma rede neural é efetivamente descobrir os pesos que serão utilizados para trazer o resultado de uma previsão**, este processo chamamos de perceptron de uma camada

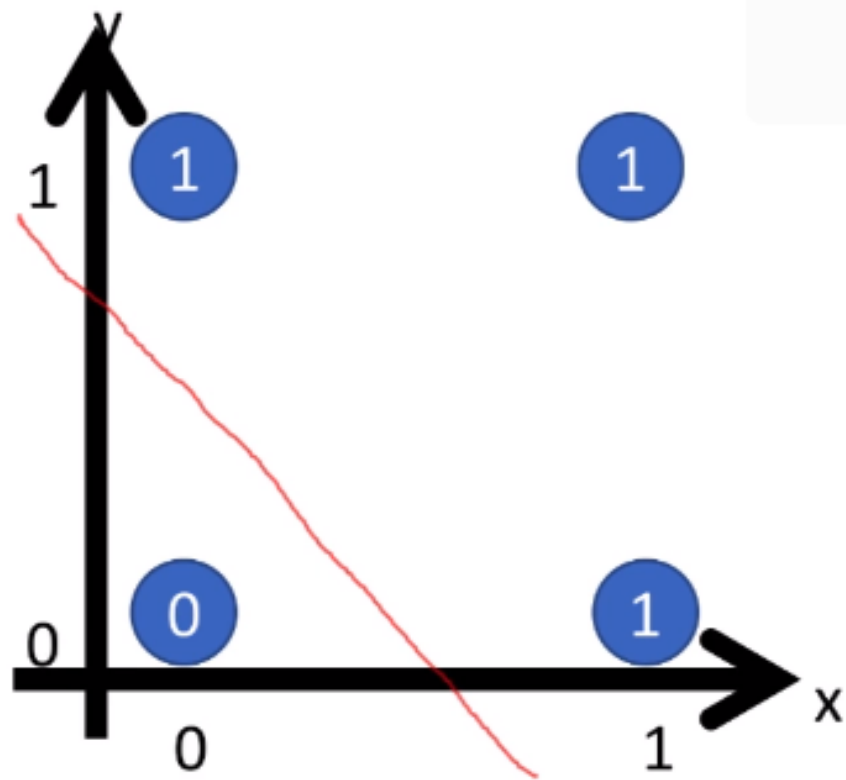
- ▼ Redes de multicamadas - Função de soma e função de ativação
- Operador AND



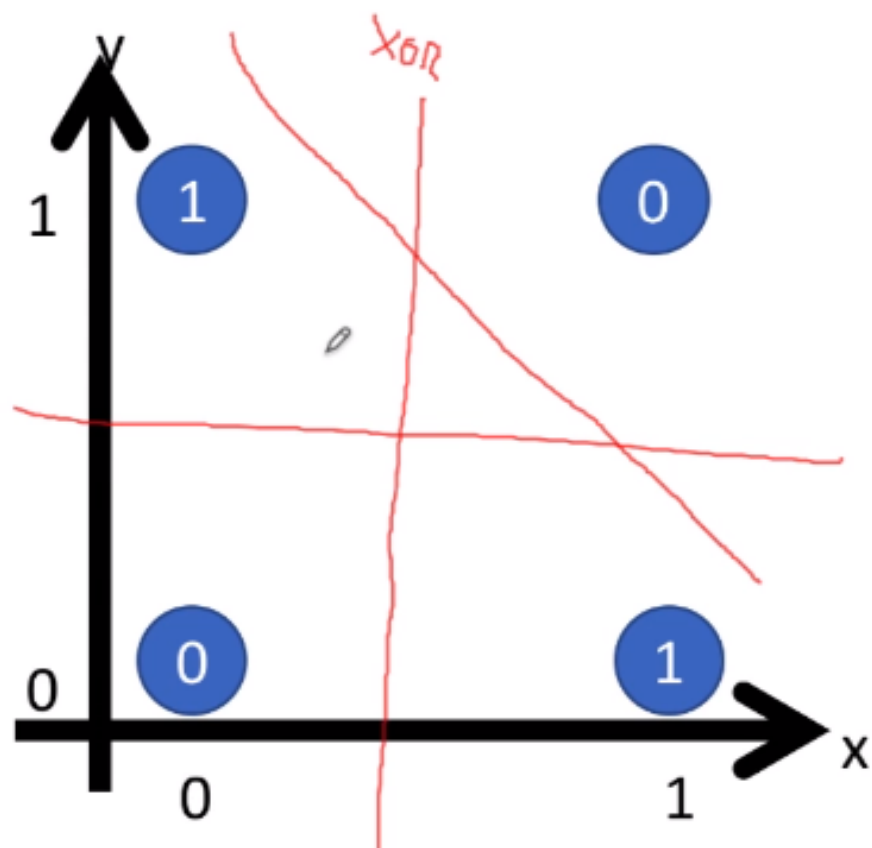
Operador XOR



Operador OR



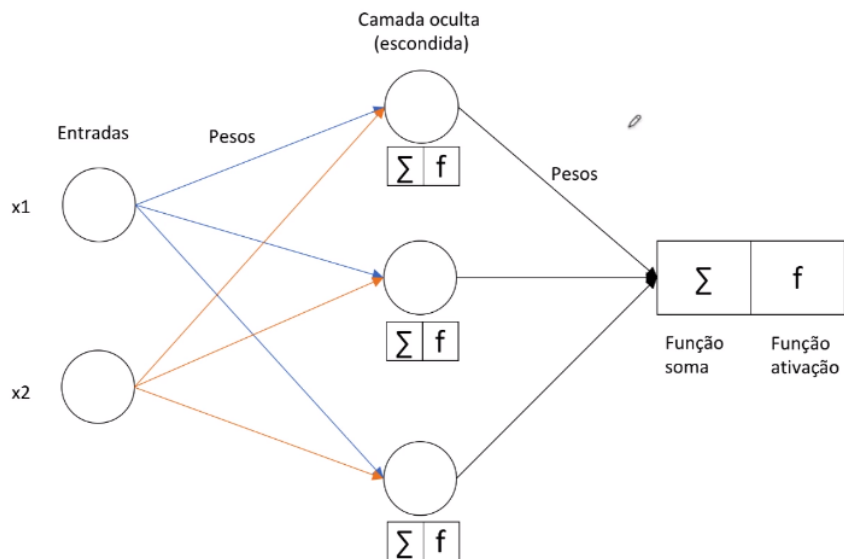
Problema não linearmente separáveis:



Ou seja, não é possível dividirmos os valores 1 e 0 no gráfico, impossibilitando encontrar os pesos.

Redes multicamadas:

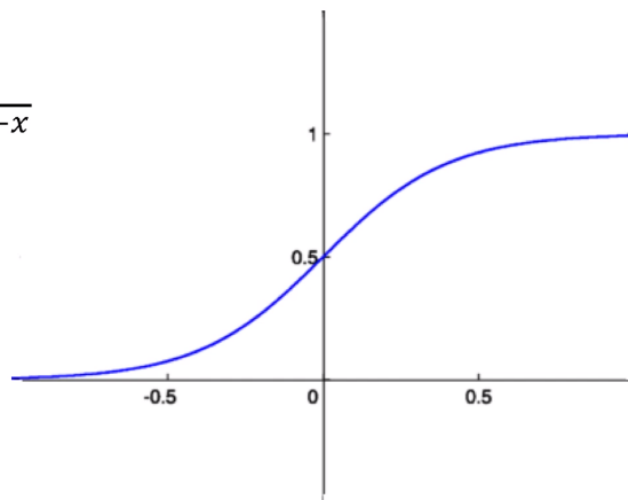




Função de ativação(Sigmoide)

Sigmoid (função sigmoide)

$$y = \frac{1}{1 + e^{-x}}$$



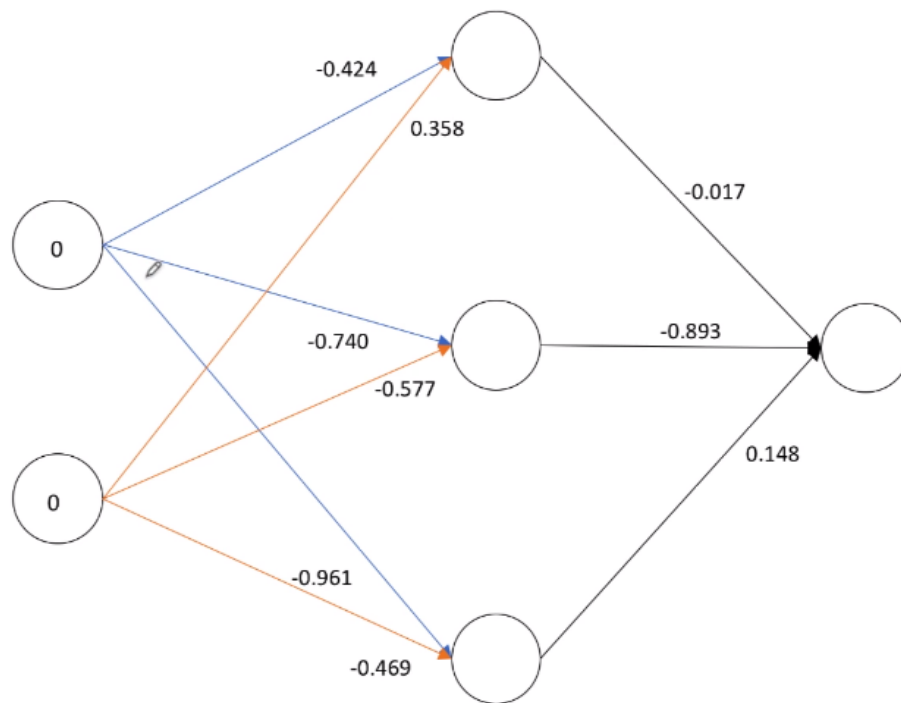
Valores entre 0 e 1.

Se valor de X for alto o valor será aproximadamente 1

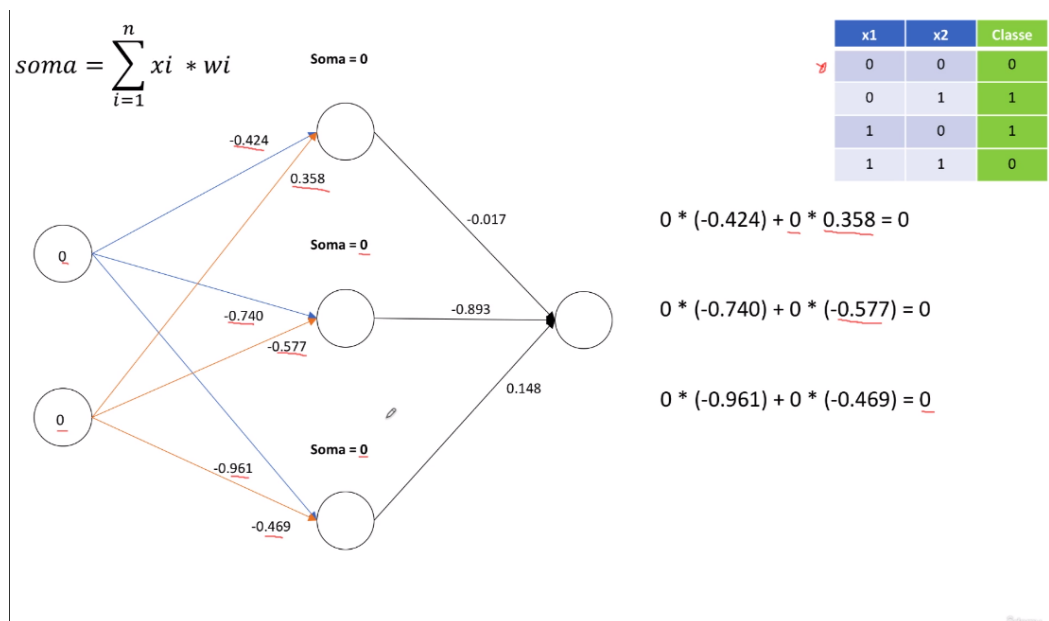
Se valor de X for pequeno o valor será aproximadamente 0

**Não retorna valores negativos**

Cálculo de valor XOR

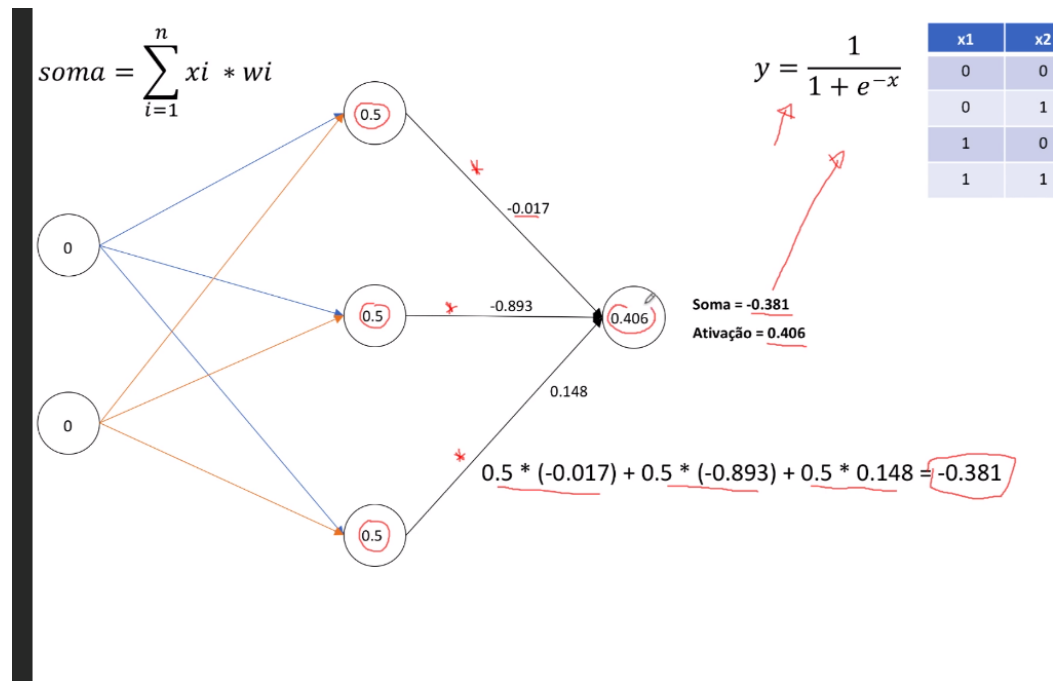


Pesos são gerados aleatoriamente (Padrão de uma inicialização de rede neural)



Aplicando a função de ativação nos valores de soma 0, retornamos em cada um o valor de 0,5

Ao termos resposta do retorno, devemos aplicar novamente a função de soma para junção da conclusão deste neurônio:



Sendo a resposta deste cálculo inicial o valor de 0,406, porém diferente de nossa tabela:

x1	x2	Classe
0	0	0

Logo, demonstrando que temos um erro.

Isto ocorreu com todos os valor da tabela

#### ▼ Rede de multicamadas - Cálculo de erro

Algoritmo mais simples para erro:

- Algoritmo mais simples
  - erro = respostaCorreta – respostaCalculada

x1	x2	Classe	Calculado	Erro
0	0	0	<del>0.406</del>	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

Média absoluta = 0.49

Ao pegarmos valores anteriores que estávamos tratando temos seguinte resultado, ficando com a média de erro 0,49.

Sendo assim, há um acerto de 51%.

#### ▼ Descida do gradiente

Gradiente = Calcular a derivada parcial para mover para a direção do gradiente.

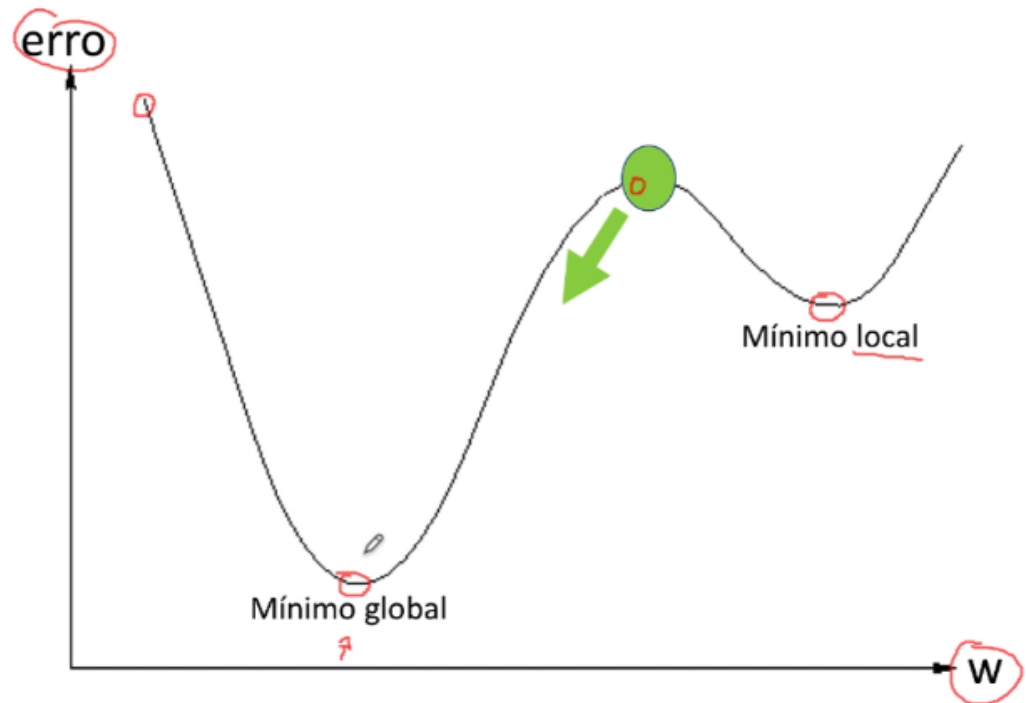
Semelhante a rolagem de uma bola em um pote, ele inicia nas bordas e vão descendo até encontrar ponto mínimo.

Min  $C(w_1, w_2, w_3, \dots, w_n)$  - Loss Function

Cálculo do gradiente

Encontrar a combinação de pesos que o erro é o menor possível

Gradiente é calculado para saber quanto ajustar os pesos



Minimo local = Local do qual algoritmo pode ou não cair, do qual evidencia uma melhoria no erro, porém, que não é os melhores pesos

Minimo Global = Melhor local para menor erro

Gradiente (derivada)

$$y = \frac{1}{1 + e^{-x}}$$



$$d = \hat{y} * (1 - y)$$

▼ Cálculo do parâmetro delta

$$\text{Soma} = -0.274$$

$$\text{Ativação} = 0.432$$

$$\text{Erro} = 1 - 0.432 = 0.568$$

$$\text{Derivada ativação (sigmoide)} = 0.245$$

$$\text{DeltaSaída} = 0.568 * 0.245 = 0.139$$

Etapas para cálculo para validação de qual lado o peso deverá aumentar para encontrar a descida do gradiente:

1. Função de ativação
2. Derivada da função
3. Delta
4. Gradiente

#### ▼ Ajuste dos pesos com backproagation

$$peso_{n+1} = (peso_n * \text{momento}) + (entrada * delta * \text{taxa de aprendizagem})$$

Momento é utilizado afim de aumentar o peso e por consequência acelerar o processo de encontrar o mínimo global. Ele é muito utilizado na decida do gradiente estocástica

Taxa de aprendizagem = 0.3

Momento = 1

Entrada x delta

0.032

0.022

0.021

$$Peso_{n+1} = (\text{peso}_n * \text{momento}) + (\text{entrada} * \text{delta} * \text{taxa de aprendizagem})$$

$$(-0.017 * 1) + 0.032 * 0.3 = \underline{-0.007}$$

$$(-0.893 * 1) + 0.022 * 0.3 = \underline{-0.886}$$

#### ▼ Bias

O Bias é um acréscimo de neurônio do qual realiza a ligação nos resultados da ligação dos neurônios e do resultado final.

Valores diferentes mesmo se todas as entradas forem zero.

Havendo um valor zero, ao haver acréscimo do Bias, evita de haver valores zerados

Muda a saída com unidade de Bias

#### ▼ Erro

Algoritmo mais simples

$$\text{Erro} = \text{RespostaCorreta} - \text{respostaCalculada}$$

MSE (Mean Square error)

$$MSE = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

x1	x2	Classe	Calculado	Erro
0	0	0	0.406	$(0 - 0.406)^2 = 0.164$
0	1	1	0.432	$(1 - 0.432)^2 = 0.322$
1	0	1	0.437	$(1 - 0.437)^2 = 0.316$
1	1	0	0.458	$(0 - 0.458)^2 = 0.209$

Após deter resultado, é realizado somatório dos registros e por fim a divisão por N afim de encontrar o valor do erro:

$$\text{Soma} = \underline{1.011}$$

$$\text{MSE} = 1.011 / 4 = \boxed{0.252}$$

RMSE(Root Mean Square Error)

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (f_i - o_i)^2}$$

Unica mudança para MSE é a raiz quadrada adicionada ao fim

$$\text{RMSE} = 0.501$$

Batch Gradient Descent (BGD)

Pegar registro por registro calculando erro e atualizar os pesos.



História do crédito	Dívida	Garantias	Renda anual	Risco
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

Primeiro carrega tudo para assim realizar processo de ajuste do peso

### Stochastic Gradient Descent (SGD)

Faz processo linha a linha

Ajuda a prevenir mínimos locais(superfícies não convexas)

Mais rápido(Não precisa carregar todos os dados em memória)

### Mini Batch Gradient Descent

Escolhe um número de registros para rodar e atualizar os pesos

### Parametros extras

Learning rate (Taxa de aprendizagem)

Batch size ( Tamanho do lote)

Epochs (épocas)

### ▼ Funções de ativação (I)

#### ▼ STEP

Utilizado para problemas linearmente separáveis

## Retorno de 0 e 1

```
import numpy as np
# Transfer function

def stepFunction(soma):
    if soma >= 1:
        return 1
    return 0

teste = stepFunction(-1)

print(teste)
```

### ▼ Sigmoid

Função bastante utilizada para probabilidade (Resultado final em uma rede neural, classificação de problemas binários)

```
def sigmoidFunction(soma):
    return 1 / (1+ np.exp(-soma))

teste = sigmoidFunction(10)

print(teste)
```

### ▼ Hyperbolic Tangent (função tangente hiperbólica)

```
def tanhFunction(soma):
    return (np.exp(soma) - np.exp(-soma)) / (np.exp(soma) + np.exp(-soma))

teste = tanhFunction(-0.358)
print(teste)
```

### ▼ Funções de ativação (II)

#### ▼ ReLu(Rectified linear units)

$Y = \max(0, x)$

Valores  $\geq 0$  (Sem valor máximo)

Utilizado quando houverem muitas camadas

#### ▼ Linear Function

Sem curvas, somente uma linha cruzando.

Muito utilizado em valores de regressão

Previsão de preço de carros

```
def linearFunction(soma):  
    return soma
```

#### ▼ Softmax

Utilizada em DeepLearning

Problemas com mais de 2 classes

Ex:

Probabilidade de algo ser verde, azul, roxo, vermelho.

$$Y = \frac{e(x)}{\sum e(x)}$$

Utilizado em redes neurais convulsionais.

```
def softmaxFunction(x):  
    ex = np.exp(x)  
    return ex / ex.sum()  
  
valores = [5.0, 2.0, 1.3]
```

```
print(softmaxFunction(valores))
```

O ex da fórmula acima é o exponencial, enquanto variável  $x$  trás um vetor de vários números


#### ▼ Módulo 4

##### ▼ Base de dados breast cancer

##### ▼ Local para adquirir datasets:

UCI Machine Learning Repository


Discover datasets around the world!

 <https://archive.ics.uci.edu>

##### ▼ Documentação Keras

Keras documentation: Layer weight initializers

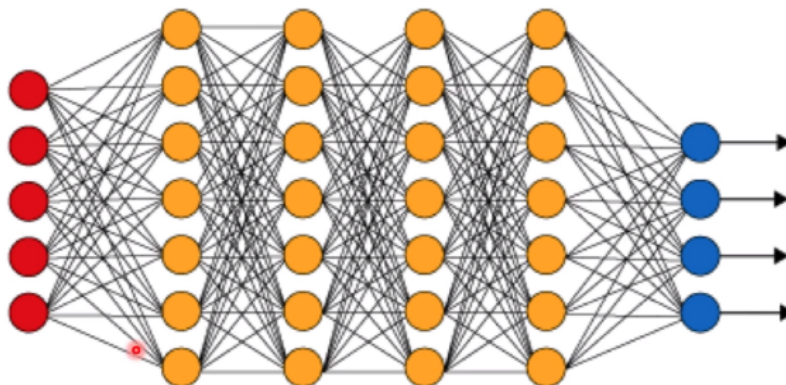
Keras documentation

 <https://keras.io/initializers/>



##### ▼ No Python (Criação)

```
from keras.layers import Dense
```



Código a seguir possibilita a criação de uma rede neural de 30 pontos oculos, que se transformam em 16 e que por sua vez retornam 1 valor:

```
import pandas as pd

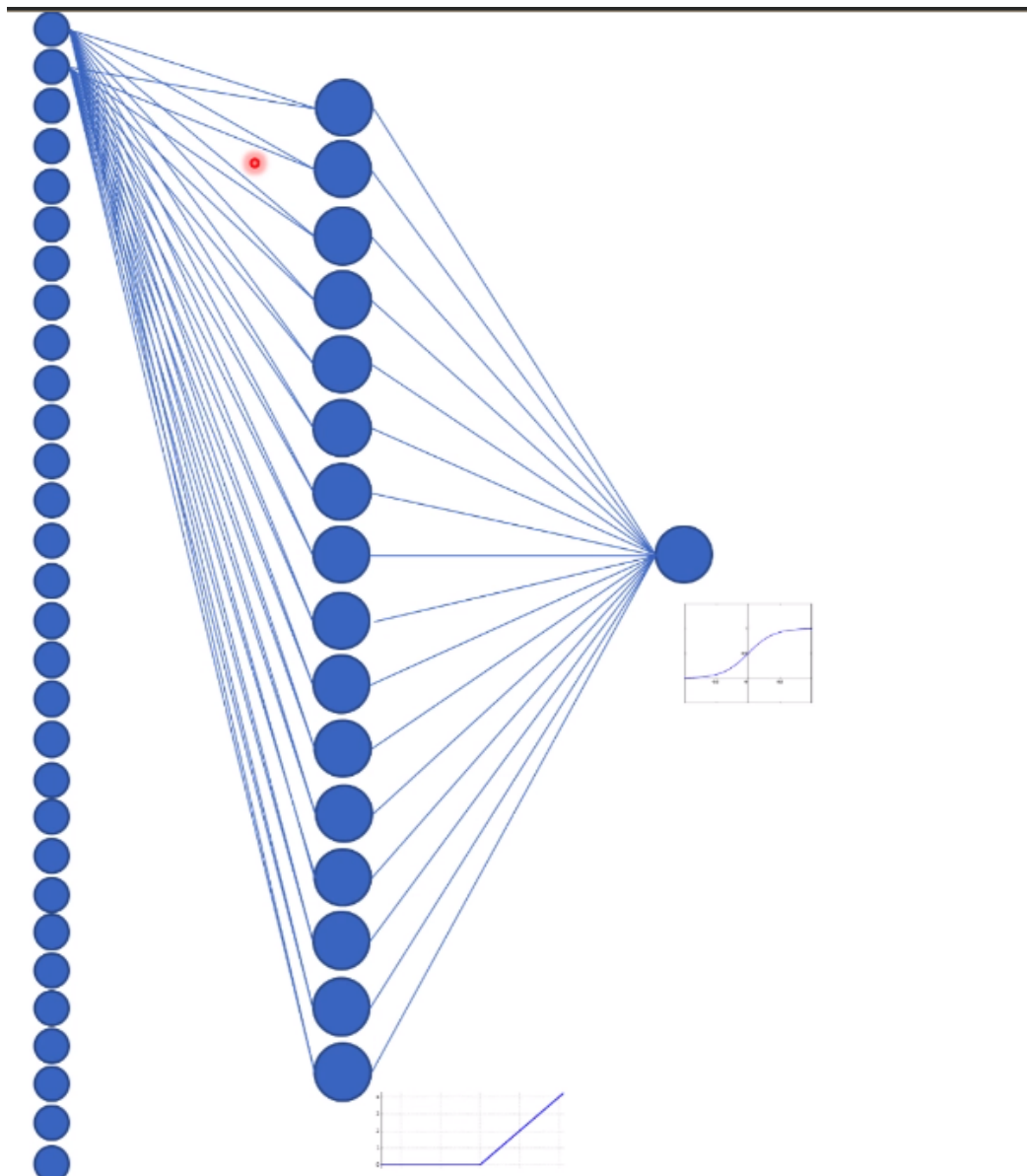
# import numpy as np
from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential
from keras.layers import Dense

previsores = pd.read_csv("entradas_breast.csv")
classe = pd.read_csv("saidas_breast.csv")

treinadoresTreinamento, previsoresTest, classeTreinamento = \
    train_test_split(previsores, classe, test_size=0.2, random_state=1)

len(previsores.columns)
classificador = Sequential()
classificador.add(
    Dense(
        units=len(previsores.columns) + 1 / 2,
        activation="relu",
        kernel_initializer="random_uniform",
        input_dim=len(previsores.columns),
    )
)

classificador.add(
    Dense(
        units=1,
        activation='sigmoid' # retorno de valor 0 e 1
    )
)
```



#### ▼ Conferencia de precisão

```
previsoes = (previsoes > 0.5)
previsao = accuracy_score(classeTeste, previsoes)
```

Utilizando o `accuracy_score` do `sklearn.metrics` é possível constatar qual foi a previsão dos seus dados teste.

```
✓ previsao ...
0.9230769230769231
```

Já através do `confusion_matrix` é possível extrair a matriz de confusão destes dados:

```
matriz = confusion_matrix(classeTeste, previsoes)
matriz
```

```
✓ matriz = confusion_matrix(classeTeste, previsoes) ...
array([[54,  8],
       [ 3, 78]], dtype=int64)
```

Também podemos realizar a validação dos resultados através do `evaluate`:

```
resultado = classificador.evaluate(previsoresTest, c
resultado
```

```
des...
[0.49356523156166077, 0.9230769276618958]
```

Este que retorna a taxa de acerto de cada um dos 2

#### ▼ MK1

```
import pandas as pd

# import numpy as np
from sklearn.model_selection import train_test_split

# import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix, accuracy_score
```

```

previsores = pd.read_csv("entradas_breast.csv")
classe = pd.read_csv("saidas_breast.csv")

previsoresTreinamento, previsoresTest, classeTreinam
    train_test_split(previsores, classe, test_size=0
)

len(previsores.columns)
classificador = Sequential()
classificador.add(
    Dense(
        units=len(previsores.columns) + 1 / 2,
        activation="relu",
        kernel_initializer="random_uniform",
        input_dim=len(previsores.columns),
    )
)

classificador.add(
    Dense(
        units=1,
        activation="sigmoid", # retorno de valor 0
    )
)

classificador.compile(
    optimizer="adam",
    # otimizador que melhor se encaixa nas mais div
    loss="binary_crossentropy",
    metrics=["binary_accuracy"],
)
# encaixar info
classificador.fit(previsoresTreinamento, classeTrein

previsoes = classificador.predict(previsoresTest)

```



```
previsoes = previsoes > 0.5
previsao = accuracy_score(classeTeste, previsoes)
previsao
matriz = confusion_matrix(classeTeste, previsoes)
matriz

resultado = classificador.evaluate(previsoresTest, c

resultado
```

#### ▼ MK2 a partir de otimizadores

```
otimizador = keras.optimizers.legacy.Adam(lr=0.001,

classificador.compile(
    optimizer=otimizador,
    # otimizador que melhor se encaixa nas mais div
    loss="binary_crossentropy",
    metrics=["binary_accuracy"],
```

A partir dos otimizadores, foi possível construir um meio de indicar parâmetros específicos para que o classificador realizasse o aprendizado de um modo diferente do indicado. Para caso em questão, criamos uma variável que recebe o formato do otimizador legado Adam, damos os atributos de lr (o quanto seu ML vai percorrer por descida no gráfico), decay (o quanto o lr vai decair em cada epoch) e clipvalue (que possibilita que seus valores não fujam do ponto mínimo). A partir disso chamamos a variável através do método compile.

#### ▼ Visualização de pesos

```
pesos0 = classificador.layers[0].get_weights()
```

A partir do código acima conseguimos validar os pesos definidos em cada epoch dos nossos neurônios.

Essa que trás como padrão os pesos padrões referentes aos pesos dos neuronios e do bias que trás como padrão na geração do nosso código.

```
pesos1 = classificador.layers[1].get_weights()
```

#### ▼ Avaliação de algoritmos

O conteúdo dos dados de treinamento afetam o quão bem seus dados estarão preparados para os testes.

K-fold Cross Validation(Quebrar base de dados em diversas partes)

Uma base de dados 4×4

Separasse em 4 retângulos com 4 itens.

Inicia-se separando primeiro retângulo para teste e as demais treinamento

Na segunda vez que rodar, retângulo seguinte será teste e demais treinamento.

...

Ao concluir sua base de dados estará preparada, tornando sua base de dados bem mais preparada que separação de 20/80

Utiliza-se normalmente o K = 10, sendo o padrão de pesquisadores.

#### ▼ Validação cruzada - Aplicação

Para primeiro passo da aplicação de uma validação cruzada, realizo o processo de criação de uma função que realiza geração de uma rede neural:

```
def criarRede():  
    classificador = Sequential()  
    classificador.add(  
        Dense(  
            units=16,  
            activation="relu",  
            kernel_initializer="random_uniform",  
            input_dim=30,
```

```

        )
    )
    classificador.add(
        Dense(
            units=16,
            activation="relu",
            kernel_initializer="random_uniform",
        )
    )
    classificador.add(Dense(units=1, activation="sigmoid"))
    otimizador = keras.optimizers.Adam(
        learning_rate=0.001, weight_decay=0.0001, clipvalue=1.0
    )
    classificador.compile(
        optimizer=otimizador, loss="binary_crossentropy",
        metrics=["binary_accuracy"]
    )
    return classificador

```

A partir de lá conseguimos complementar com seguintes comandos:

```

classificador = KerasClassifier(model=criarRede, epochs=100,
                                batch_size=32)

resultados = cross_val_score(
    estimator=classificador,
    X=previsores,
    y=classe,
    cv=10, # quantas vezes será rodado teste
    scoring="precision", # Substitua "loss" pela métrica desejada
)

```

Primeiro deles, é o classificador, ele chama o método `kerasClassifier` com os parâmetros `Model` (a função que criamos, a qual define o modelo da nossa rede neural), `epochs` (que define a quantidade de épocas que serão rodadas no nosso código) e `batch_size` (que define quantas repetições serão rodadas).

Feito isto, é chamado uma outra variável, que recebe o resultado de cada uma das repetições.

Após isto, foi permitido gerar 2 variáveis que detém dos valores de resultado da nossa rede neural:

```
media = resultados.mean()  
desvio = resultados.std()
```

#### ▼ Overfitting e Underfitting

##### Underfitting

"Subestimar problema"

"Rodar um jogo com jogadores da reserva"

Problemas mais complexos dos quais estão tentando ser resolvidos com soluções mais simples

Resultados ruins na base de treinamento

##### Overfitting

Bons resultados na base de treinamento

Se adapta muito aos dados da base de treino

Resultados ruins na base de teste

Muito específico

Memorização

Erros na variação de novas instâncias

#### ▼ Dropout afim de diminuir o Overfitting

"Mata neuronios" afim de tornar eles inutilizaveis.

Processo é realizado de 20%/30% dos neuronios.

```
classificador.add(Dropout(0.2))
```

Para acréscimo, chamar módulo add com atributo Dropout deste modo, ele estará "melhorando" os resultados evitando o Overfitting

#### ▼ Tuning

Método para identificar através de testes, qual melhores parametros para utilizar durante o desenvolvimento da rede neural.

```
grid_search = GridSearchCV(estimator = classificador
                           param_grid = parametros,
                           scoring = 'accuracy',
                           cv = 5)
grid_search = grid_search.fit(previsores, classe)
melhores_parametros = grid_search.best_params_
melhor_precisao = grid_search.best_score_
```

Como exemplo acima, construímos uma chamada de função, que utilizava seguinte função para estruturação:

```
def criarRede(optimizer, loss, kernel_initializer, activation):
    classificador = Sequential()
    classificador.add(
        Dense(
            units=neurons,
            activation=activation,
            kernel_initializer=kernel_initializer,
            input_dim=30,
        )
    )
    classificador.add(Dropout(0.2))
    classificador.add(
        Dense(
            units=neurons,
            activation=activation,
            kernel_initializer=kernel_initializer,
        )
    )
    classificador.add(Dropout(0.2))
    classificador.add(Dense(units=1, activation="sigmoid"))

    classificador.compile(
        optimizer=optimizer, loss=loss,
        metrics=["binary_accuracy"]
    )
```

```
)
return classificador
```

```
classificador = KerasClassifier(build_fn = criarRede
parametros = {'batch_size':[10,30],
               'epochs':[50, 100],
               'model__optimizer':['adam', 'sgd'],
               'loss':['binary_crossentropy', 'hinge']
               'model__kernel_initializer':['random_u
               'model__activation':['relu'],
               'model__neurons': [16, 8]}}
```

A partir dela, foi possível identificar qual seriam os melhores parâmetros a se utilizar na rede neural.

#### ▼ Classificação de somente um registro

Através dos parametros alcançados a partir do comando rodado no módulo anterior, conseguimos identificar quais melhores métodos para nossa rede neural:

```
classificador = Sequential()

classificador.add(Dense(units=8, activation = 'relu'
                        kernel_initializer= 'normal'
classificador.add(Dropout(0.2))
classificador.add(Dense(units=8, activation = 'relu'
                        kernel_initializer='normal',
classificador.add(Dropout(0.2))
classificador.add(Dense(units = 1, activation = 'sig
classificador.compile(optimizer = 'adam', loss = "bi
                        metrics = ['binary_accuracy'])
```

Fazemos então inicialização dos classificadores

```
classificador.fit(previsores, classe, batch_size = 10)
```

Rodamos nosso script para aprendizado da rede neural

Criei uma pequena estrutura para criar dados aleatórios de teste

```
novo = ([])  
for n in range(30):  
    juntar = random.random()*100  
    juntar = round(juntar, 2)  
    novo = np.append(novo, juntar)  
novo = np.array([novo])
```

#### ▼ Salvar rede neural

```
classificadorJson = classificador.to_json()
```

Direcione dados do classificador para uma variável com parametro .to\_json()

```
with open("classificadorBrest.json", "w") as json_file:  
    json_file.write(classificadorJson)
```

Após isto, realize processo de carregamento dos dados do json para um arquivo de fato

```
classificador.save_weights("classificador_breast.h5")
```

Grave os pesos utilizado a partir do seguinte código acima.

#### ▼ Carregar rede neural

E após isto realize processo de geração de um teste para identificação.

```
novo = ([])  
for n in range(30):
```

```
juntar = random()*20
juntar = round(juntar, 2)
novo = np.append(novo, juntar)
novo = np.array([novo])
novo
```

Feito teste do carregamento da rede, importe a rede de teste que deseja fazer os testes:

```
previsores = pd.read_csv("entradas_breast.csv")
classe = pd.read_csv("saidas_breast.csv")
```

E por fim rode a compilação:

```
classificador.compile(optimizer = 'adam', loss = "binary_crossentropy",
                      metrics = ['binary_accuracy'])
```

Rodando a compilação, basta identificar os valores da validação:

```
resultado = classificador.evaluate(previsores, classe)
```

## ▼ Módulo 5

### ▼ Base de dados iris

Importação da base e geração de um dataframe:

```
import pandas as pd
base = pd.read_csv("iris.csv")
```

Separação em variáveis do tipo array com os dados deste dataframe

```
# Criação de uma variavel somente para previsores e out
# Iloc trás determinadas informações do nosso dataframe
# No caso abaixo, desejo todas as linhas com as 4 primeiras colunas
previsores = base.iloc[:, 0:4].values
```

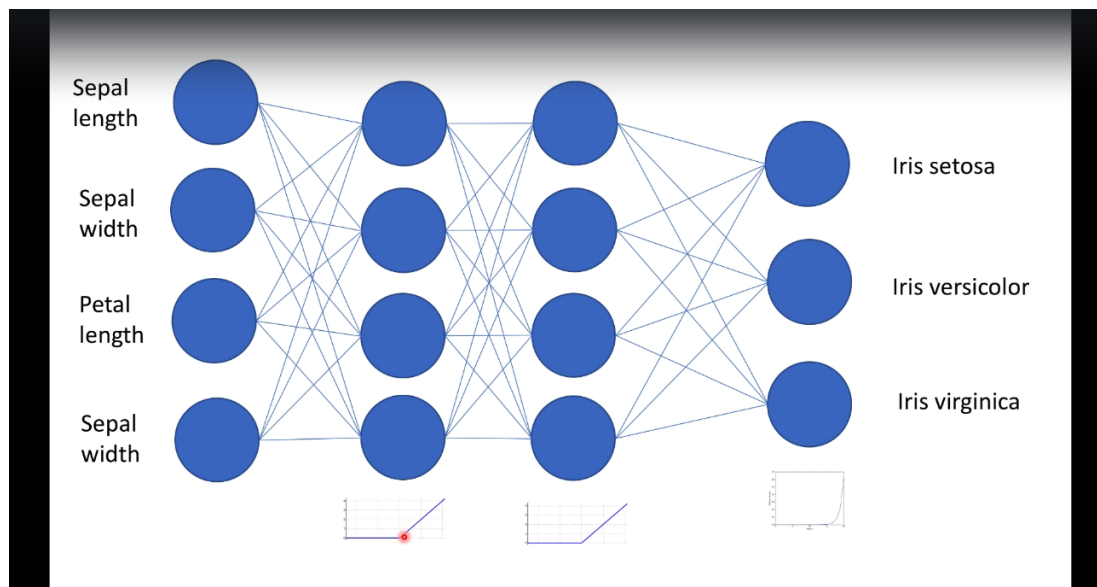


```
# Para esta database, a partir da 4 coluna trás as resp
classe = base.iloc[:,4].values
```

Split dos dados para treino e teste:

```
from sklearn.model_selection import train_test_split
# Modelo de particionamento de uma database

previsoresTreinamento, previsoresTeste, classeTreinamen
```



```
import pandas as pd

base = pd.read_csv("iris.csv")

# Criação de uma variavel somente para previsores e out
# Iloc trás determinadas informações do nosso dataframe
# No caso abaixo, desejo todas as linhas com as 4 prime
previsores = base.iloc[:, 0:4].values

# Para esta database, a partir da 4 coluna trás as resp
classe = base.iloc[:, 4].values
```

```

previsores

from sklearn.model_selection import train_test_split
# Modelo de particionamento de uma database

previsoresTreinamento, previsoresTeste, classeTreinamen
    train_test_split(previsores, classe_dummy, test_size=
)

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

# Construção da estrutura da rede neural

classificador = Sequential()
# Units = Quantidade de colunas + quantidade de saídas(
# input_dim quantidade de atributos quantos neuronios n
classificador.add(Dense(units=4, activation="relu", inp

# Segunda camada oculta
classificador.add(Dense(units=4, activation="relu"))

# Por não se tratar de uma estrutura binária, existem m
# Sendo assim, neurônio de saída representa 3 unidades
# Problema de mais de duas classes, função correta é a
# Gera probabilidade de ser cada uma das opções
classificador.add(Dense(units=3, activation="softmax"))

# kullback_leibler_divergence também serve para loss fu
classificador.compile(
    optimizer="adam", loss="categorical_crossentropy",
)

# Treinamento com problemas de mais de 2 classes não pe

# Atributo classe contem 3 informações. Para ajuste, é
from sklearn.preprocessing import LabelEncoder

```

```

# Transforma os 3 atributos em 3 valores diferentes
labelencoder = LabelEncoder()
classe = labelencoder.fit_transform(classe)

# Se retorno no ultimo neuronico cair seguintes paramet
# Conseguimos definir o que de fato serão
# iris setosa 1 0 0
# iris virginica 0 1 0
# iris versicolor 0 0 1

from keras.utils import to_categorical

classe_dummy = to_categorical(classe)

classificador.fit(previsoresTreinamento, classeTreiname

resultado = classificador.evaluate(previsoresTeste, cla

resultado

previsoes = classificador.predict(previsoresTeste)
previsoes = (previsoes > 0.5)

import numpy as np
classeTeste2 = [np.argmax(t) for t in classeTeste]

previsoes2 = [np.argmax(t) for t in previsoes]

from sklearn.metrics import confusion_matrix
matriz = confusion_matrix(previsoes2, classeTeste2)

matriz

```

Através do código acima, pude alcançar pudemos realizar processo de desenvolvimento da rede neural para identificação de uma estrutura multivaloral

#### ▼ Problemas

Ao tentar rodar através do método crossValidation, mesmo utilizando Vs com versão atualizadas da biblioteca, ou colab com versões padrão, não pude conseguir rodar devidamente, gerando criticas nos tamanhos. Ao identificar repasse das respostas, professores indicaram para realizar processo de downgrade na versão do keras e tensor flow.

```
import pandas as pd
import keras_utils
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
base = pd.read_csv("iris.csv")

# Criação de uma variavel somente para previsores e out
# Iloc trás determinadas informações do nosso dataframe
# No caso abaixo, desejo todas as linhas com as 4 prime
previsores = base.iloc[:, 0:4].values

# Para esta database, a partir da 4 coluna trás as resp
classe = base.iloc[:, 4].values
previsores

# Modelo de particionamento de uma database
from sklearn.model_selection import train_test_split
previsoresTreinamento, previsoresTeste, classeTreinamen
    train_test_split(previsores, classe_dummy, test_siz
)

from keras.models import Sequential
from keras.layers import Dense
# Construção da estrutura da rede neural
# Atributo classe contem 3 informações. Para ajuste, é
from sklearn.preprocessing import LabelEncoder
```

```

# Transforma os 3 atributos em 3 valores diferentes
labelencoder = LabelEncoder()
classe = labelencoder.fit_transform(classe)

# Se retorno no ultimo neuronico cair seguintes paramet
# Conseguimos definir o que de fato serão
# iris setosa 1 0 0
# iris virginica 0 1 0
# iris versicolor 0 0 1

from keras.utils import to_categorical
classe_dummy = to_categorical(classe)

def criar_rede():
    classificador = Sequential()
    classificador.add(Dense(units=4, activation="relu",
    classificador.add(Dense(units=4, activation="relu")
    classificador.add(Dense(units=3, activation="softmax")
    classificador.compile(
        optimizer="adam", loss="categorical_crossentropy"
    )
    return classificador

classificador = KerasClassifier(build_fn=criar_rede,
                                epochs=1000,
                                batch_size=10)

resultados = cross_val_score(estimator = classificador,
                              X = previsores,
                              y= classe,
                              cv = 10,
                              scoring = 'accuracy')

media = resultados.mean()
desvio = resultados.std()

```

Acima criação do comando

```
-----
File c:\Users\leofe\Documents\LABIA12 - Prática Redes Neurais (II)\estrutura rede neural (Iris)\crossValidation.py:55
49 return classificador
51 classificador = KerasClassifier(build_model_rede,
52                                epochs=1000,
53                                batch_size=10)
----> 55 resultados = cross_val_score(estimator = classificador,
56                                x = preditores,
57                                y = classe,
58                                cv = 10,
59                                scoring = 'accuracy')
61 media = resultados.mean()
62 desvio = resultados.std()

File ~\Anaconda\Local\Packages\PythonSoftwareFoundation.Python.3.11.qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\utils\_param_validation.py:213, in validate_params.<locals>.<decorator.<locals>.wrapper(*args, **kwargs)
207 try:
208     with config_context(
209         skip_parameter_validation=(
210             prefer_skip_nested_validation or global_skip_validation
211         )
212     ):
--> 213         return func(*args, **kwargs)
214 except InvalidParameterError as e:
215     # when the function is just a wrapper around an estimator, we allow
...
raise e.with_traceback(filtered_tb) from None
File "c:\Users\leofe\Anaconda\Local\Packages\PythonSoftwareFoundation.Python.3.11.qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\keras\src\backend\tensorflow\nn.py", line 553, in categorical_crossentropy
raise ValueError(
ValueError: Arguments 'target' and 'output' must have the same shape. Received: target.shape=(None, 1), output.shape=(None, 3)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.
```

Segue critica gerada.

## ▼ Módulo 6

### Analise carro

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

base = pd.read_csv("autos.csv", encoding="ISO-8859-1")
# Exclusão das colunas que não serão utilizadas
base = base.drop("dateCrawled", axis=1)
base = base.drop("dateCreated", axis=1)
base = base.drop("nrOfPictures", axis=1)
base = base.drop("postalCode", axis=1)
base = base.drop("lastSeen", axis=1)

base["name"].value_counts()
# Nome detém de diversos nomes diferentes
# Dos quais não tem relação e apoio nos valores
base = base.drop("name", axis=1)

# Informação não é válida para verificação
# pela quantidade de atributos
base["seller"].value_counts()
base = base.drop("seller", axis=1)
```

```

# Valores de oferta referentes a leilão não
# devem ser analisados juntos
base["offerType"].value_counts()
base = base.drop("offerType", axis=1)

# Primeiro passo é sempre a redução de ruído
# afim de tornar nosso algoritmo mais eficiente

# Não faz sentido os valores abaixo de 10 euros para carro
i1 = base.loc[base.price <= 10]

# Validação valor médio de venda
base.price.mean()

# "Exclusão" dos valores do dataframe
base = base[base.price > 10]

# valor acima de 350000
i2 = base.loc[base.price > 350000]

base = base.loc[base.price < 350000]

# Ajuste dos valores NaN
base.loc[pd.isnull(base["vehicleType"])]
base["vehicleType"].value_counts() # limousine

base.loc[pd.isnull(base["gearbox"])]
base["gearbox"].value_counts() # manuell

base.loc[pd.isnull(base["model"])]
base["model"].value_counts() # golf

base.loc[pd.isnull(base["fuelType"])]
base["fuelType"].value_counts() # benzin

base.loc[pd.isnull(base["notRepairedDamage"])]

```

```

base["notRepairedDamage"].value_counts() # nein

valores = {
    "vehicleType": "limousine",
    "gearbox": "manuell",
    "model": "golf",
    "fuelType": "benzin",
    "notRepairedDamage": "nein",
}

base = base.fillna(value=valores)

previsores = base.iloc[:, 1:13].values
precoReal = base.iloc[:, 0].values

# Transformação de campos categóricos

labelencoderPrevisores = LabelEncoder()

previsores[:, 0] = labelencoderPrevisores.fit_transform(p
previsores[:, 1] = labelencoderPrevisores.fit_transform(p
previsores[:, 3] = labelencoderPrevisores.fit_transform(p
previsores[:, 5] = labelencoderPrevisores.fit_transform(p
previsores[:, 8] = labelencoderPrevisores.fit_transform(p
previsores[:, 9] = labelencoderPrevisores.fit_transform(p
previsores[:, 10] = labelencoderPrevisores.fit_transform(

# Criação da classe Dummy

# 0 -> 0 0 0
# 2 -> 0 1 0
# 3 -> 0 0 1

# Identificação do tipo específico, afim de categorizar el

onehotencoder = ColumnTransformer(
    transformers=[("OneHot", OneHotEncoder(), [0, 1, 3, 5,

```



```

        remainder="passthrough",
    )
    previsores = onehotencoder.fit_transform(previsores).toarray()

    regressor = Sequential()
    regressor.add(Dense(units=158, activation="relu", input_dim=158))
    regressor.add(Dense(units=158, activation="relu"))
    # Sigmoid 0 -> 1
    # Mais um retorno Softmax
    # linear não fará nada
    # como queremos somente valor somente ela retornará isto
    regressor.add(Dense(units=1, activation="linear"))

    # 2 carros base de dados
    # primeira previsão 1000(valor real) - 900(valor previsto)
    # Segunda previsão 2000(valor real) - 2100(valor previsto)
    # Erro total = 100 + 100 (independente se for negativo somamos)
    # Por conta disso utilizamos o mean_absolute_error
    regressor.compile(loss = 'mean_absolute_error', optimizer = 'adam',
                      metrics = ['mean_absolute_error'])

    regressor.fit(previsores, precoReal, batch_size = 300, epochs = 100)

    previsoes = regressor.predict(previsores)

    precoReal.mean()
    previsoes.mean()

```

## CrossVal

```

import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import cross_val_score

```

```

from scikeras.wrappers import KerasRegressor

base = pd.read_csv("autos.csv", encoding="ISO-8859-1")
# Exclusão das colunas que não serão utilizadas
base = base.drop("dateCrawled", axis=1)
base = base.drop("dateCreated", axis=1)
base = base.drop("nrOfPictures", axis=1)
base = base.drop("postalCode", axis=1)
base = base.drop("lastSeen", axis=1)

base["name"].value_counts()
# Nome detém de diversos nomes diferentes
# Dos quais não tem relação e apoio nos valores
base = base.drop("name", axis=1)

# Informação não é válida para verificação
# pela quantidade de atributos
base["seller"].value_counts()
base = base.drop("seller", axis=1)

# Valores de oferta referentes a leilão não
# devem ser analisados juntos
base["offerType"].value_counts()
base = base.drop("offerType", axis=1)

# Primeiro passo é sempre a redução de ruído
# afim de tornar nosso algoritmo mais eficiente

# Não faz sentido os valores abaixo de 10 euros para carro
i1 = base.loc[base.price <= 10]

# Validação valor médio de venda
base.price.mean()

# "Exclusão" dos valores do dataframe

```

```

base = base[base.price > 10]

# valor acima de 350000
i2 = base.loc[base.price > 350000]

base = base.loc[base.price < 350000]

# Ajuste dos valores NaN
base.loc[pd.isnull(base["vehicleType"])]
base["vehicleType"].value_counts() # limousine

base.loc[pd.isnull(base["gearbox"])]
base["gearbox"].value_counts() # manuell

base.loc[pd.isnull(base["model"])]
base["model"].value_counts() # golf

base.loc[pd.isnull(base["fuelType"])]
base["fuelType"].value_counts() # benzin

base.loc[pd.isnull(base["notRepairedDamage"])]
base["notRepairedDamage"].value_counts() # nein

valores = {
    "vehicleType": "limousine",
    "gearbox": "manuell",
    "model": "golf",
    "fuelType": "benzin",
    "notRepairedDamage": "nein",
}

base = base.fillna(value=valores)

previsores = base.iloc[:, 1:13].values
precoReal = base.iloc[:, 0].values

# Transformação de campos categóricos

```

```

labelenconderPrevisores = LabelEncoder()

previsores[:, 0] = labelenconderPrevisores.fit_transform(p
previsores[:, 1] = labelenconderPrevisores.fit_transform(p
previsores[:, 3] = labelenconderPrevisores.fit_transform(p
previsores[:, 5] = labelenconderPrevisores.fit_transform(p
previsores[:, 8] = labelenconderPrevisores.fit_transform(p
previsores[:, 9] = labelenconderPrevisores.fit_transform(p
previsores[:, 10] = labelenconderPrevisores.fit_transform(

# Criação da classe Dummy

# 0 -> 0 0 0
# 2 -> 0 1 0
# 3 -> 0 0 1

# Identificação do tipo específico, afim de categorizar el

onehotencoder = ColumnTransformer(
    transformers=[("OneHot", OneHotEncoder(), [0, 1, 3, 5,
    remainder="passthrough",
    )
previsores = onehotencoder.fit_transform(previsores).toarray()

def criarRede():
    regressor = Sequential()

    regressor.add(Dense(units=158, activation="relu", input_dim=input_dim))
    regressor.add(Dense(units=158, activation="relu"))
    regressor.add(Dense(units=1, activation="linear"))
    regressor.compile(
        loss="mean_absolute_error", optimizer="adam", metrics=['accuracy']
    )
    return regressor

input_dim = previsores.shape[1]

```

```

regressor = KerasRegressor(build_fn=criarRede, epochs=100,

resultados = cross_val_score(
    estimator=regressor,
    X=previsores,
    y=precoReal,
    cv=10,
    scoring="neg_mean_absolute_error",
)
media = resultados.mean()
desvio = resultados.std()

```

## ▼ Módulo 7

```

import pandas as pd
from keras.layers import Dense, Dropout, Activation, Input
from keras.models import Model
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

base = pd.read_csv(
    "C:/Users/leofe/Documents/LAMIA/Prática Redes Neurais2.
)

# Identificação de campos que não serão uteis

base = base.drop("Other_Sales", axis=1)
base = base.drop("Global_Sales", axis=1)

# Existe outro campo com informações do Developer já
base = base.drop("Developer", axis=1)

# Tratamento valores faltantes
# Processo será realizado a partir da exclusão dos valores

base = base.dropna(axis=0)

```

```

base = base.loc[base["NA_Sales"] > 1]
base = base.loc[base["EU_Sales"] > 1]
base = base.loc[base["NA_Sales"] > 1]

base["Name"].value_counts
nomeJogos = base.Name
base = base.drop("Name", axis=1)

previsores = base.iloc[:, [0, 1, 2, 3, 7, 8, 9, 10, 11]].values

vendaNa = base.iloc[:, 4].values
vendaEu = base.iloc[:, 5].values
vendaJp = base.iloc[:, 6].values

labelencoder = LabelEncoder()
# Ajuste dos valores string em valores numéricos
previsores[0]
previsores[:, 0] = labelencoder.fit_transform(previsores[:, 0])
previsores[:, 2] = labelencoder.fit_transform(previsores[:, 2])
previsores[:, 3] = labelencoder.fit_transform(previsores[:, 3])
previsores[:, 8] = labelencoder.fit_transform(previsores[:, 8])

# Acrescimo de colunas afim de transformar generos em valores numéricos
onehotencoder = ColumnTransformer(
    transformers=[("OneHot", OneHotEncoder(), [0, 2, 3, 8])],
    remainder="passthrough",
)
previsores = onehotencoder.fit_transform(previsores).toarray()

camadaEntrada = Input(shape=(65,))
# Como não vamos utilizar um método sequencial é necessário o uso de Input
# das camadas após fechamento dos parenteses
camadaOculta1 = Dense(
    units=32,
    activation="sigmoid",
)(camadaEntrada)

```

```

camadaOculta2 = Dense(units=32, activation="sigmoid")(camadaOculta1)
camadaSaida1 = Dense(units=1, activation="linear")(camadaOculta2)
camadaSaida2 = Dense(units=1, activation="linear")(camadaOculta2)
camadaSaida3 = Dense(units=1, activation="linear")(camadaOculta2)

regressor = Model(
    inputs=camadaEntrada, outputs=[camadaSaida1, camadaSaida2, camadaSaida3]
)
regressor.compile(optimizer="adam", loss="mse")
regressor.fit(previsores, [vendaNa, vendaEu, vendaJp], epochs=100,
               batch_size=100)

previsaoNa, previsaoEu, previsaoJp = regressor.predict(previsores)
previsaoNa

```