```
!pip install tensorflow
!pip install scikit-learn
!pip install pandas
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
         Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
        Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
        Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)

Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
         Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
        Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
       Requirement already satisfied: Iloclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
        Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0) Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
         Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
       Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
        Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
        Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.44.0) Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.9.2)
        Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8) Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13.0)
        Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.0) Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10) Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
        Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30) Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.7)
        Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorf]
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.0.4)
         Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow
        Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (3.0.0) Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)
        Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1 Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
        Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4) Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
         Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
        Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0) Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
        Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.26.4) Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
         Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
        Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

Curso: Redes Neurais e Deep Learning

Prof. Denilson Alves Pereira https://sites.google.com/ufla.br/denilsonpereira/ Departamento de Ciência da Computação - Instituto de Ciências Exatas e Tecnológicas - Universidade Federal de Lavras

Atividade Prática 01

Instruções:

- 1. Siga os passos indicados em cada célula abaixo para completar a atividade.
- 2. Você deve inserir código somente entre as linhas marcadas com **INICIE O CÓDIGO AQUI** e **TERMINE O CÓDIGO AQUI**. Há uma indicação de quantas linhas de código são necessárias.
- 3. Em alguns pontos, confira o resultado esperado conforme marcado com SAÍDA ESPERADA.

Tempo estimado para execução: 1 hora

Versão: Junho. 2021

O Problema a ser Resolvido

O objetivo da atividade é elaborar uma rede neural para predizer se um paciente tem ou não diabetes, com base nas medidas diagnósticas contidas no *dataset* disponível em https://www.kaggle.com/uciml/pima-indians-diabetes-database.

Os dados são de pacientes do sexo feminino, com pelo menos 20 anos de idade. Os atributos das condições médicas incluem o número de gestações que a paciente teve, seu IMC, nível de insulina, idade e outros. A classe a ser predita é o atributo "Outcome", cujos valores são 0 (não tem diabetes) ou 1 (tem diabetes). Portanto, é um problema de classificação binária.

Você vai praticar as seguintes habilidades:

- Efetuar o pré-processamento dos dados, separando-os em conjuntos de treino e teste.
- Configurar uma rede neural simples para um problema de classificação binária.

Pacotes

```
import numpy as np # package for scientific computing
import tensorflow as tf # package for numerical computation using data flow graphs
from tensorflow import keras # package for deep learning
import pandas as pd # package for working with structured data
```

Pré-Processamentos dos Dados de Treino e de Teste

```
data = pd.read_csv('diabetes.csv')
data.head() # display dataset first lines
        Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age Outcome
     0
                                       72
                                                     35
                                                               0 33.6
                                                                                         0.627 50
     1
                 1
                        85
                                       66
                                                     29
                                                               0 26.6
                                                                                         0.351 31
                                                                                                          0
     2
                 8
                        183
                                       64
                                                      0
                                                              0 23.3
                                                                                         0.672 32
                                                                                                          1
                         89
                                       66
                                                     23
                                                              94 28.1
     3
                                                                                         0.167 21
                                                                                                          0
     4
                 0
                        137
                                       40
                                                     35
                                                             168 43.1
                                                                                         2.288 33
```

```
# Separate the class from other attributes
X = data.drop("Outcome", axis=1)
Y = data["Outcome"]
```

Read dataset

Documentação de train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

A função divide os dados em partições de treino e teste, de acordo com a proporção especificada pelo parâmetro test_size.

O parâmetro random_state é usado para deixar os resultados reproduzíveis para fins de avaliação do exercício.

```
# Preparing the dataset for training and test
from sklearn.model_selection import train_test_split
train_set_X, test_set_X, train_set_Y, test_set_Y = train_test_split(X, Y, test_size=0.20, random_state=7)
```

Padronize os atributos usando a média e a variância dos dados

Dica: use a função fit_transform: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

```
# Standardize features by removing the mean and scaling to unit variance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

### INICIE 0 CÓDIGO AQUI ### (2 linhas de código)
train_set_X = scaler.fit_transform(train_set_X)
test_set_X = scaler.fit_transform(test_set_X)
### TERMINE 0 CÓDIGO AQUI ###

# Checking
```

SAÍDA ESPERADA:

train set X:

 $[[\ 0.35483802\ -0.370418\ 0.16624635\ 1.40032403\ -0.01279543\ 0.49863797\ -0.61786951\ 0.01064085]$

 $[-0.54794048 - 0.55620696 \ 0.9024924 \ 0.96548604 \ 0.40069886 \ 1.70739891 \ -1.0345765 \ -0.86049025]]$

test_set_X:

 $\llbracket [-0.83168001 \ -1.14367855 \ -0.39723824 \ -0.57849223 \ -0.36848171 \ -0.47210798 \ 0.22800925 \ -0.84284936 \rrbracket$

 $[\ 0.8647269\ 1.8633592\ 0.67258938\ 0.00976179\ 0.72124393\ 0.58740671\ 0.24525553\ 1.28629259]]$

Obtenha o número de atributos e o número de exemplos de treinamento

Dica: use a função *shape*: https://numpy.org/devdocs/reference/generated/numpy.shape.html

```
### INICIE 0 CÓDIGO AQUI ### (2 linhas de código)
n = train_set_X.shape[1]# number of attributes
m =train_set_X.shape[0]# number of training examples
### TERMINE 0 CÓDIGO AQUI ###

print ("Number of attributes: n = " + str(n))
print ("Number of training examples: m = " + str(m))
print ("Train set X shape: " + str(train_set_X.shape))
print ("Train set Y shape: " + str(train_set_Y.shape))
print ("Test set X shape: " + str(test_set_X.shape))
print ("Test set Y shape: " + str(test_set_Y.shape))
```

```
Number of attributes: n = 8

Number of training examples: m = 614

Train set X shape: (614, 8)

Train set Y shape: (614,)

Test set X shape: (154, 8)

Test set Y shape: (154,)
```

SAÍDA ESPERADA:

Number of attributes: n = 8

Number of training examples: m = 614

Train set X shape: (614, 8) Train set Y shape: (614,) Test set X shape: (154, 8) Test set Y shape: (154,)

Definição do Modelo

Crie um modelo em Keras com a seguinte configuração:

- Camada de entrada: no formato dos dados de entrada do problema
- Camada 1: 3 neurônios, função de ativação *Tanh*
- Camada 2: 5 neurônios, função de ativação *Tanh*
- Camada 3: 3 neurônios, função de ativação Tanh
- Camada 4 (saída): 1 neurônio, função de ativação Sigmoid

Dica 1: use a classe Model: https://keras.io/api/models/model/

Dica 2: veja as funções de ativação disponíveis: https://keras.io/api/layers/activations/

```
### INICIE O CÓDIGO AQUI ### (6 linhas de código)
inputs = keras.Input(shape=(train_set_X.shape[1],))
x = keras.layers.Dense(units=3, activation='tanh')(inputs)
x = keras.layers.Dense(units=5, activation='tanh')(x)
x = keras.layers.Dense(units=3, activation='tanh')(x)
outputs = keras.layers.Dense(units=1, activation='sigmoid')(x)
model = keras.Model(inputs=inputs, outputs=outputs)
### TERMINE O CÓDIGO AQUI ###
```

```
# Checking
processed_data = model(train_set_X)
print(processed_data.shape)
```

→ (614, 1)

SAÍDA ESPERADA:

(614, 1)

Prints a summary of the network, showing its architecture and parameters.
model.summary()

~ ...

→ Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 8)	0
dense (Dense)	(None, 3)	27
dense_1 (Dense)	(None, 5)	20
dense_2 (Dense)	(None, 3)	18
dense_3 (Dense)	(None, 1)	4

Total params: 69 (276.00 B)
Trainable params: 69 (276.00 B)
Non-trainable params: 0 (0 00 R)

SAÍDA ESPERADA:

Confira a configuração da rede e o total de parâmetros = 69

Compilação do Modelo

Compile o model usando os seguintes parâmetros:

- Função de perda: mean_absolute_error
- Otimizador: RMSprop
- Métricas: accuracy, Precision, Recall

Dica 1: use a função *compile*: https://keras.io/api/models/model_training_apis/

 $\label{losses} \mbox{Dica 2: relação de funções de perda: } \mbox{$\frac{https://www.tensorflow.org/api_docs/python/tf/keras/losses}$\\$

 $\label{linear_potential} \mbox{Dica 3: relação de otimizadores: } \mbox{$\frac{https://www.tensorflow.org/api_docs/python/tf/keras/optimizers}$\\$

Dica 4: relação de métricas: https://keras.io/api/metrics/

```
### INICIE 0 CÓDIGO AQUI ### (1 linha de código)
model.compile(loss='mean_absolute_error', optimizer='RMSprop', metrics=['accuracy', keras.metrics.Precision(), keras.metrics.Recall()])
### TERMINE 0 CÓDIGO AQUI ###
```

Treinamento do Modelo

Ajusta o modelo aos dados de treinamento. Devem ser fornecidos os dados de treinamento, o número de épocas (iterações) e o tamanho do lote (batch). Uma época é composta por uma única passagem por todos os exemplos do conjunto de treino. O tamanho do lote define o número de amostras (exemplos) a serem consideradas pelo modelo antes de atualizar os pesos. Assim, uma época é composta por um ou mais lotes.

Efetue o treinamento do modelo usando os seguintes parâmetros:

- Tamanho do lote: 64
- Número de épocas: 1000

Dica: use a função fit: https://keras.io/api/models/model_training_apis/

```
### TERMINE O CÓDIGO AQUI ###
print(history.history) # print per-epoch timeseries of metrics values
    Epoch 1/1000
     10/10 ———— Epoch 2/1000
                               - 2s 4ms/step - accuracy: 0.5385 - loss: 0.4956 - precision: 0.3345 - recall: 0.4739
     10/10
                               - 0s 3ms/step - accuracy: 0.6356 - loss: 0.4799 - precision: 0.4501 - recall: 0.4589
     Epoch 3/1000
     10/10
                                0s 3ms/step - accuracy: 0.6383 - loss: 0.4768 - precision: 0.4732 - recall: 0.3964
     Epoch 4/1000
     10/10
                                0s 4ms/step - accuracy: 0.6501 - loss: 0.4681 - precision: 0.5298 - recall: 0.3978
     Epoch 5/1000
                               - 0s 3ms/step - accuracy: 0.6829 - loss: 0.4588 - precision: 0.5486 - recall: 0.3692
     10/10
     Epoch 6/1000
                               - 0s 3ms/step - accuracy: 0.7164 - loss: 0.4454 - precision: 0.5940 - recall: 0.4160
     10/10
     Epoch 7/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7035 - loss: 0.4448 - precision: 0.6182 - recall: 0.3703
     Epoch 8/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.6825 - loss: 0.4372 - precision: 0.5348 - recall: 0.3322
     Epoch 9/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7078 - loss: 0.4258 - precision: 0.6384 - recall: 0.3663
     Epoch 10/1000
     10/10
                                0s 4ms/step - accuracy: 0.7033 - loss: 0.4166 - precision: 0.6001 - recall: 0.3454
     Epoch 11/1000
10/10 ———
                               - Os 3ms/step - accuracy: 0.6798 - loss: 0.4190 - precision: 0.5838 - recall: 0.3196
     Epoch 12/1000
     10/10
                               - 0s 4ms/step - accuracy: 0.7093 - loss: 0.4010 - precision: 0.6052 - recall: 0.3201
     Epoch 13/1000
     10/10
                               - 0s 4ms/step - accuracy: 0.7109 - loss: 0.3985 - precision: 0.6730 - recall: 0.3501
     Epoch 14/1000
     10/10
                               - 0s 4ms/step - accuracy: 0.7263 - loss: 0.3806 - precision: 0.6554 - recall: 0.3286
     Epoch 15/1000
     10/10
                                0s 4ms/step - accuracy: 0.7140 - loss: 0.3819 - precision: 0.6684 - recall: 0.3612
     Epoch 16/1000
     10/10
                                0s 3ms/step - accuracy: 0.7138 - loss: 0.3783 - precision: 0.6881 - recall: 0.3203
     Epoch 17/1000
                               - Os 4ms/step - accuracy: 0.7187 - loss: 0.3758 - precision: 0.7289 - recall: 0.3722
     10/10
     Epoch 18/1000
                               - 0s 3ms/step - accuracy: 0.7132 - loss: 0.3696 - precision: 0.6545 - recall: 0.3766
     10/10
     Epoch 19/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7339 - loss: 0.3561 - precision: 0.6866 - recall: 0.3852
     Epoch 20/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7154 - loss: 0.3651 - precision: 0.7022 - recall: 0.3780
     Epoch 21/1000
     10/10
                                0s 4ms/step - accuracy: 0.7344 - loss: 0.3495 - precision: 0.7168 - recall: 0.3886
     Epoch 22/1000
     10/10
                                0s 4ms/step - accuracy: 0.7392 - loss: 0.3389 - precision: 0.7252 - recall: 0.3879
     Epoch 23/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7066 - loss: 0.3565 - precision: 0.6666 - recall: 0.3828
     Epoch 24/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7401 - loss: 0.3392 - precision: 0.7356 - recall: 0.4106
     Epoch 25/1000
     10/10
                               - 0s 4ms/step - accuracy: 0.7610 - loss: 0.3207 - precision: 0.7598 - recall: 0.4472
     Epoch 26/1000
     10/10
                               - 0s 3ms/step - accuracy: 0.7465 - loss: 0.3287 - precision: 0.6839 - recall: 0.4547
     Epoch 27/1000
     10/10
                               - 0s 4ms/step - accuracy: 0.7657 - loss: 0.3135 - precision: 0.7283 - recall: 0.4924
     Epoch 28/1000
     10/10
                                0s 3ms/step - accuracy: 0.7512 - loss: 0.3159 - precision: 0.7201 - recall: 0.4774
     Epoch 29/1000
```

SAÍDA ESPERADA:

Na época 1000, tem-se o seguinte resultado (aproximado):

loss: 0.1938 - accuracy: 0.8062 - precision: 0.7644 - recall: 0.6303

Avaliação do Modelo

Avalie o desempenho da rede no conjunto de teste.

INICIE O CÓDIGO AQUI ### (1 linha de código)

history = model.fit(train_set_X, train_set_Y, batch_size=64, epochs=1000)

Dica: use a função *evaluate*: https://keras.io/api/models/model_training_apis/

```
### INICIE 0 CÓDIGO AQUI ### (1 linha de código)
loss, acc, prec, rec = model.evaluate(test_set_X, test_set_Y)
### TERMINE 0 CÓDIGO AQUI ###
print("Loss: %.2f" % loss, "\nAccuracy: %.2f" % acc, "\nPrecision: %.2f" % prec, "\nRecall: %.2f" % rec)

$\frac{5}{5} \frac{1}{5} \frac{1}{5
```

Predição

Precision: 0.76 Recall: 0.61

Apresente a predição do conjunto de teste.

Dica: use a função *predict*: https://keras.io/api/models/model_training_apis/

```
### INICIE O CÓDIGO AQUI ### (1 linha de código)
predictions = model.predict(test_set_X)
```

SAÍDA ESPERADA:

Valores aproximados:

TERMINE O CÓDIGO AQUI

Desafio

Modifique a configuração da sua rede e/ou os parâmetros dos métodos com o objetivo de melhorar os resultados das métricas no conjunto de teste. Experimente várias opções.

Adicione abaixo a nova seguência de código que alcançou o melhor resultado.

```
### INICIE 0 CÓDIGO AQUI ### (várias linhas de código / várias células)

## Não consegui alcançar uma accuracy maior, porém precisão sim. 0 modelo foi o seguinte:
inputs = keras.Input(shape=(train_set_X.shape[1],))
x = keras.layers.Dense(units=3, activation='leaky_relu')(inputs)
x = keras.layers.Dense(units=5, activation='leaky_relu')(x)
x = keras.layers.Dense(units=3, activation='leaky_relu')(x)
outputs = keras.layers.Dense(units=1, activation='sigmoid')(x)

# Resultado:
# Loss: 0.22
# Accuracy: 0.78
# Precision: 0.81
# Recall: 0.53

### TERMINE 0 CÓDIGO AQUI ###
```

Fim

Parabéns! Você efetuou todos os passos para criar uma rede neural com várias camadas para um problema de classificação binária.