AULA 14
INTRODUÇÃO AO TENSORFLOW

1. Objetivos

- Introduzir a ferramenta de desenvolvimento TensorFlow.
- Apresentar o princípio básico de funcionamento do TensorFlow.
- Apresentar as estruturas de dados usadas pelo TensorFlow.
- Mostrar semelhanças e diferenças entre Numpy e TensorFlow.
- Apresentar o modo de operação eager do TensorFlow.
- Apresentar o TensorBoard

2. Introdução

- TensorFlow é a ferramenta de desenvolvimento de redes neurais mais utilizada. Informações e manual podem ser encontrados em https://www.tensorflow.org/.
- TensorFlow foi desenvolvido pelo Google e está sempre sendo atualizado ⇒ deve-se tomar cuidado com as inúmeras versões existentes.
- A utilização do TensorFlow sem o Keras não é muito simples e exige conhecimento profundo de programação em Python.
- Existem diversas formas de utilizar o TensorFlow:
  - No caso de redes neurais, a forma mais fácil é usar o Keras do TensorFlow, como visto nas aulas anteriores;
  - Porém, muitos programas são feitos com o TensorFlow sem usar o Keras, assim, em é importante conhecer um pouco as outras formas de usar o TensorFlow.
- TensorFlow ⇒ realiza operações com tensores e é muito parecido com a biblioteca Numpy.
- O TensorFlow possui inúmeras funções para implementar cálculos com tensores, redes neurais e muitos outros tipos de cálculos ⇒ muito difícil conhecer tudo e para fazer cálculos mais complexos deve-se procurar ajuda na internet.

3. Princípio básico de funcionamento

- Em princípio, o processo de desenvolver um programa usando o TensorFlow envolve duas etapas:
  - Construir um Gráfico Computacional ⇒ **Etapa de construção**;
  - Executar o Gráfico Computacional ⇒ **Etapa de execução**.
- **Etapa de construção** ⇒ quando se define as variáveis e os cálculos a serem realizados.

  Na etapa de construção é construído um Gráfico Computacional que consiste de um fluxograma com todos os cálculos e variáveis definidas.
- **Etapa de execução ⇒** quando se executa o Gráfico Computacional definido na etapa de construção.

Construção de um Gráfico Computacional

- Um Gráfico Computacional consiste de uma série de operações estruturadas como um fluxograma com nós e conexões entre esses nós.
- Cada nó do fluxograma recebe variáveis como entradas e gera outra variável como saída.
- Na Figura 1 é mostrado um exemplo simples de operações usando o TensorFlow e seu Gráfico Computacional equivalente. Esse Gráfico consiste de 3 nós: `a`, `b` e `c`.

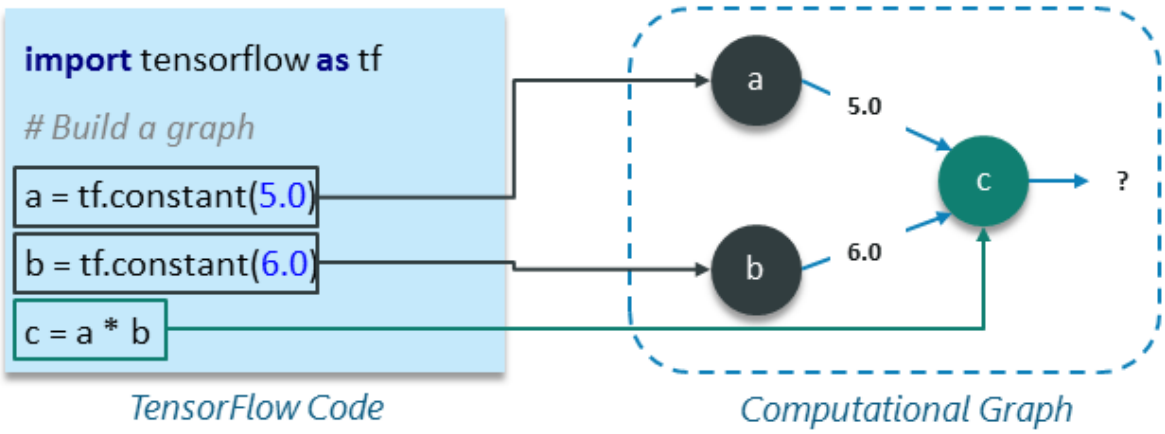


Figura 1. Exemplo de operações definidas com o TensorFlow e seu respectivo Gráfico Computacional (https://www.edureka.co/blog/tensorflow-tutorial/).

- Nós de constantes são usados para armazenar valores constantes. Esses nós não possuem entradas e geram os valores armazenados como saída. No exemplo da Figura 1, `a` e `b` são nós de constantes com valores 5 e 6 respectivamente.

- O nó `c` representa a operação de multiplicação dos nós de constantes `a` e `b` ⇒ assim, executar esse Gráfico resulta na saída do nó `c` que é multiplicação de `a` por `b`.

➢ Basicamente, um Gráfico Computacional consiste de uma forma alternativa de representar cálculos matemáticos realizados em um programa desenvolvido com o TensorFlow.

➢ Para que serve um Gráfico Computacional?

- As operações atribuídas a cada nó do gráfico podem ser realizadas em paralelo e com isso melhora o desempenho computacional da execução dos cálculos;
- Mas o mais importante é que o uso do Gráfico Computacional permite criar um modelo simbólico dos cálculos, para posteriormente, se for necessário, calcular os gradientes das funções definidas no gráfico ⇒ isso é fundamental no treinamento das RNAs.

➢ **Na criação do Gráfico Computacional não se calcula nada e também nenhum valor é atribuído às variáveis, somente se define as operações especificadas no programa.**

➢ Operações definidas em um programa com TensorFlow geram um gráfico computacional que não tem valores numéricos até serem avaliados explicitamente.

Execução de um Gráfico Computacional

➢ No quadro a seguir é apresentada a criação do Gráfico Computacional, mostrado na Figura 1, com o TensorFlow.

```
import tensorflow as tf

# Construção do gráfico
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b

# Imprime variável c
print(c)
```

```
Tensor("mul_2:0", shape=(), dtype=float32)
```

- Observe que o comando `print(c)` apresenta somente o tipo de variável e sua dimensão, mas não o seu valor, pois nesse momento a variável `c` ainda não foi calculada ⇒ o cálculo numérico da variável `c` é realizado somente após a execução do Gráfico.

➢ Para calcular `c` é necessário executar o Gráfico Computacional ⇒ a execução de um Gráfico Computacional é realizada dentro de uma **Sessão do TensorFlow**.

- Uma Sessão compila o programa do Gráfico e fornece meios de executar os comandos;
- Uma Sessão é um objeto que encapsula o ambiente no qual as variáveis são calculadas numericamente quando solicitado;
- Uma Sessão armazena a informação da sequencia na qual todas as operações são realizadas e passa os resultados já calculados para a operação seguinte do fluxograma.

- No quadro a seguir é mostrado como executar o Gráfico Computacional definido no quadro anterior.

```
# Cria uma instância da classe Sessão
sess = tf.Session()

# Executa o Gráfico dentro da Sessão e armazena a variável de saída
c = sess.run(c)

# Imprime o resultado do Gráfico
print('c=', c)

# Fecha a Sessão
sess.close()
```

```
c= 30.0
```

  - Observe a sintaxe utilizada para definir e executar uma Sessão:
    - `tf.Session` é a classe de Seções do TensorFlow;
    - sess é o nome dado para a Sessão (pode ser qualquer nome);
    - `sess.run(c)` é o método que executa a Sessão e realiza os cálculos numéricos;

- O método `sess.run(c)` faz com que todas as operações definidas para calcular a variável `c` sejam executadas dentro da Sessão.

- Outra forma de criar o Gráfico Computacional do exemplo anterior e executá-lo é mostrado no quadro a seguir.

```
# Define duas constantes
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
print(c)

# Abre, executa e fecha uma Sessão
with tf.Session() as sess:
    sess.run(c)
    print(c.eval())
sess.close()
```

```
Tensor("mul_2:0", shape=(), dtype=float32)
30.0
```

  - Observe que o primeiro comando `print(c)` somente apresenta os atributos da variável, pois o cálculo numérico da variável `c` é realizado somente após a execução da Sessão.

- Pode-se realizar vários cálculos em uma única sessão, como mostrado no quadro a seguir.

```
import tensorflow as tf

# Define 3 constantes
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)

# Define os cálculos
intermed = tf.add(input2, input3)
mul = tf.multiply(input1,intermed)

# Abre e executa sessão para realizar os cálculos
with tf.Session() as sess:
    result1, result2 = sess.run([mul, intermed])
    print('Multiplicação=', result1,'-', 'Soma=', result2)
sess.close()
```

```
Multiplicação= 21.0 - Soma= 7.0
```

- Ao executar `sess.run([mul, intermed])` o TensorFlow realiza todos os cálculos definidos para determinar as variáveis `intermed` e `mul`.
- Observe que realizar cálculos com TensorFlow somente é possível se forem usados os seus operadores, que no caso são `tf.add` e `tf.multiply`.

Sessão Iterativa do TensorFlow

- Uma Sessão Iterativa do TensorFlow consiste de uma forma conveniente de manter uma Sessão aberta em um Notebook Jupyter e, assim, poder realizar inúmeros cálculos numericamente.
- Classe utilizada para Sessão Iterativa ⇒ **`tf.InteractiveSession()`**.
- No quadro a seguir é apresentado um exemplo de cálculo usando uma Sessão Iterativa e sua avaliação explícita.

```
# Abre uma Sessão iterativa
sess = tf.InteractiveSession()

# Define o Gráfico Computacional
a = np.zeros((2,2))
ta = tf.zeros((2,2))
print(a)
print(ta)

# Atribui o valor numérico à variável ta
print(ta.eval())

# Fecha a Sessão
sess.close()
```

```
[[0. 0.]
 [0. 0.]]
Tensor("zeros_1:0", shape=(2, 2), dtype=float32)
[[0. 0.]
 [0. 0.]]
```

- Observe que o cálculo numérico da variável `ta` somente é realizado com o método `eval()`.
- Qualquer cálculo numérico no TensorFlow deve ser realizado dentro de uma Sessão ⇒ nesse exemplo é utilizada uma Sessão Iterativa, `tf.InteractiveSession()`.
- O método `close()` também serve para finalizar uma Sessão interativa.

3. Constantes, dados de entrada e variáveis

- No TensorFlow constantes, dados de entrada e variáveis são usadas para representar diferentes parâmetros de uma RNA.

Constantes

- Constantes no TensorFlow servem para armazenar constantes que não variam durante a execução do programa.
- Constantes são inicializadas quando são criadas com o comando `tf.constant` e seus valores nunca mudam.
- Todos os tensores usados nos exemplos anteriores eram constantes.

Dados de entrada ("Placeholders")

- Dados de entrada para um modelo (RNA) no TensorFlow são definidos por meio de "placeholders".
- Um "placeholder" consiste na alocação e preparação de espaço, realizada na construção de um Gráfico Computacional, para futuramente receber variáveis na hora de executar o programa.
- No quadro a seguir é apresentado um exemplo de uso e aplicação de um "placeholder.

```
import tensorflow as tf

# Cria placeholders compostos por variáveis tipo float32
a = tf. placeholder(tf.float32)
b = tf. placeholder(tf.float32)

# Calcula multiplicação de a por b
c = a*b

# Inicializa uma Sessão
sess = tf.Session()

# Calcula c passando os valores a = [1, 3] e b = [2, 4]
output = sess.run(c, {a: [1,3], b: [2, 4]})
print('Multiplicação de a por b:', output)
sess.close()
```

```
Multiplicação de a por b= [ 2. 12.]
```

- Um “placeholders” não inicializa e não contém nenhum valor ou dado.
- Para execução do programa deve-se fornecer (“alimentar”) as entradas para o “placeholder”.
- Tentar executar um programa sem fornecer dados para um “placeholder” gera mensagem de erro.
- No quadro a seguir é apresentado um exemplo de outra forma de alimentar um “placeholder” com dados de entrada.

```
import tensorflow as tf

# Cria placeholders compostos por variáveis tipo float32
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

# Define cálculo
output = tf.multiply(input1, input2)

# Abre e executa sessão
with tf.Session() as sess:
    result = sess.run([output], feed_dict={input1:[7.],input2:[2.]})
    print(result)
sess.close()
```

```
[array([14.], dtype=float32)]
```

Variáveis

- Variáveis são espaços de memórias no TensorFlow que contém tensores.
- Variáveis são usadas para manter e atualizar parâmetros, ou seja, podem ser modificadas durante a execução do programa.
- Variáveis são definidas fornecendo seus valores iniciais e tipos.

- No caso de não se definir explicitamente o tipo da variável, o TensorFlow infere a partir do formato dos números usados na definição da variável.
- Valores iniciais para as variáveis são definidos quando elas são criadas.
- Variáveis podem ser inicializadas com valores constantes ou com números aleatórios.
- Pode-se dar um nome para a variável.

➢ Para inicializar as variáveis no TensorFlow deve-se usar explicitamente o método `tf.global_variables_initializer()` dentro de uma Sessão.

➢ No quadro a seguir é mostrado como definir e inicializar variáveis.

```
import tensorflow as tf

# Cria tensor W1 de uns com dimensão 2x2
W1 = tf.ones((2,2))

# Define variável W2 de dimensão 2x2 com zeros
W2 = tf.Variable(tf.zeros((2,2)), name="weights")

# Define variável R de dimensão 2x2 com números aleatórios
R = tf.Variable(tf.random_normal((2,2)), name="random_weights")

#Cria sessão
with tf.Session() as sess:
    # Executa sessão com W1
    print('w1=', sess.run(W1))

    # Inicializa variáveis
    sess.run(tf.global_variables_initializer())

    #Executa sessão com W2
    print('w2=', sess.run(W2))

    #Executa sessão com R
    print('R=', sess.run(R))
```

```
w1= [[1. 1.]
 [1. 1.]]
w2= [[0. 0.]
 [0. 0.]]
R= [[0.8929614  0.12805592]
 [1.2338506  1.4730624 ]]
```

- `W1` não precisa ser inicializado porque não é uma variável do TensorFlow.
- `W2` e `R` foram criadas como sendo variáveis e, portanto, precisam ser inicializadas antes de serem usadas.

➢ **Diferença entre variáveis e outros tipos de estruturas:**

- Variáveis no TensorFlow somente são instanciadas (criadas) quando um Gráfico é executado;
- Variáveis são mantidas após a execução de uma sessão;

- Outros tipos de variáveis são automaticamente apagados após uma sessão ser executada.

Uso de variáveis em cálculos

- No quadro a seguir é mostrado um exemplo de como definir e realizar cálculos usando o TensorFlow. Nesse programa a variável `state` é um contador que é incrementado de 1, por 3 iterações.

```
import tensorflow as tf

# Cria variável state
state = tf.Variable(0, name="counter")

# Incrementa 1 na variável state ⇒ new_value = state + 1
new_value = tf.add(state, tf.constant(1))

# Atribui novo valor na variável state ⇒ state = new_value
update = tf.assign(state, new_value)

# Abre sessão para realizar os cálculos
with tf.Session() as sess:
    # Inicializa variáveis
    sess.run(tf.global_variables_initializer())
    print('state=', sess.run(state))

    # Executa os cálculos
    for _ in range(3):
       print('state=', sess.run(update))

# Fecha sessão
sess.close()
```

```
state= 0
state= 1
state= 2
state= 3
```

- Observe que o comando `tf.assign(state, new_value)` atribui o valor de `new_value` à variável `state` e salva na varável `update`.
- Ao `executar sess.run(update)` o TensorFlow realiza todos os cálculos definidos para calcular a variável `update`.

- No quadro a seguir é mostrado como esses cálculos seriam realizados por um programa usando Python simples.

```
state = 0
print('state=', state)
for _ in range(3)
    state = state + 1
    print('state=', state)
```

4. TensorFlow versus Numpy

- TensorFlow e Numpy são muito similares ⇒ ambos operam com tensores de múltiplas dimensões.

- Vantagens e desvantagens:
  - Numpy tem a desvantagem de não oferecer métodos para criar funções de tensores e calcular suas derivadas automaticamente;
  - Numpy não tem suporte para cálculo em GPU;
  - TensorFlow tem a desvantagem de ter que definir os cálculos antes de poderem ser realizados.

- No quadro a seguir é apresentado, como exemplo, alguns cálculos realizados com Numpy e com o TensorFlow. Esses cálculos mostram algumas características do TensorFlow e suas diferenças em relação ao Numpy.

```
# Cálculo com Numpy

import numpy as np

a = np.zeros((2,2))
b = np.ones((2,2))
c = np.sum(b, axis=0)
d = np.reshape(a, (1,4))

print('b=', b)
print('c=', c)
print('Dimensão de a=', a.shape)
print('d=', d)
```

```
b= [[1. 1.]
 [1. 1.]]
c= [2. 2.]
Dimensão de a= (2, 2)
d= [[0. 0. 0. 0.]]
```

```
# Cálculo com TensorFlow

import tensorflow as tf

tf.InteractiveSession()
a = tf.zeros((2,2))
b = tf.ones((2,2))
c = tf.reduce_sum(b,reduction_indices=0)
d = tf.reshape(a, (1, 4))

print('b=' b.eval())
print('c=', c.eval())
print('Dimensão de a=', a.get_shape())
print('d=', d.eval())
```

```
b= [[1. 1.]
 [1. 1.]]
c= [2. 2.]
Dimensão de a= (2, 2)
d= [[0. 0. 0. 0.]]
```

- Observe que o TensorFlow exige abrir uma Sessão e incluir o comando (`eval`) para realizar o cálculo numérico das variáveis.

- Na Tabela 1 são apresentados alguns comandos do Numpy e seus equivalentes no TensorFlow.

Tabela 1. Alguns comandos do Numpy e seus equivalentes no TensorFlow

Numpy	TensorFlow
`a = np.zeros((2,2))`	`a = tf.zeros((2,2))`
`b = np.ones((2,2))`	`b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`c = b * 5 + 1`	`c = b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`# seleção de elementos de tensor` `a[0,0], a[:,0], a[0,:]`	`# seleção de elementos de tensor` `a[0,0], a[:,0], a[0,:]`

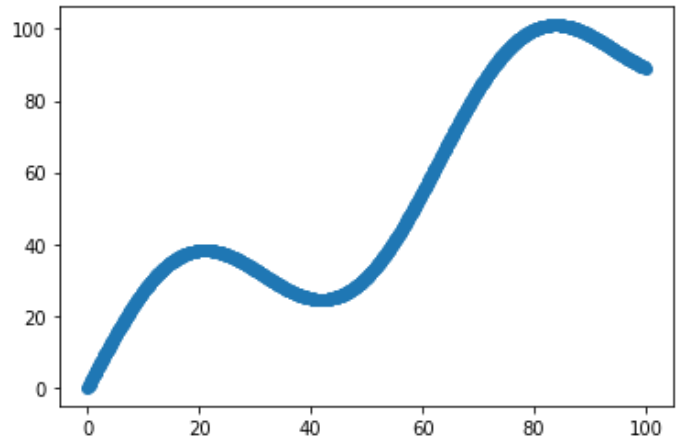
5. Exemplo de solução de problema usando TensorFlow

- Como exemplo de uso do TensorFlow é mostrado como resolver um problema de ajuste de função usando regressão linear.

- No quadro a seguir são gerados os dados do problema.

```
import numpy as np
import matplotlib.pyplot as plt

# Define dados do problema
X_data = np.arange(100, step=.1)
y_data = X_data + 20*np.sin(X_data/10)

# Dimensão dos dados de entrada e de saída
print(X_data.shape, y_data.shape)

# Gráfico dos dados
plt.scatter(X_data, y_data)
plt.show()
```

```
(1000,) (1000,)
```



- No quadro a seguir são definidos os "placeholders" para alimentar os dados de entrada.

```
import tensorflow as tf

# Define dimensão dos dados
n_samples = X_data.shape[0]

# TensorFlow é muito sensível à dimensão dos dados de entrada,
assim, é sempre bom redefinir essas dimensões sem incerteza
X_data = np.reshape(X_data, (n_samples,1))
y_data = np.reshape(y_data, (n_samples,1))
print(X_data.shape, y_data.shape)

# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(n_samples, 1))
y = tf.placeholder(tf.float32, shape=(n_samples, 1))
```

```
(1000, 1) (1000, 1)
```

  - Observe que a dimensão original dos dados de entrada era (1000,) e após o seu "acerto" ficou sendo (1000, 1).

- No quadro a seguir são definidos e inicializados os parâmetros de ajuste (`W` e `b`), definida a função de ajuste (reta) e a função de custo.

```
# Define parâmetros da regressão a serem calculados
b = tf.Variable(tf.zeros((1,)), name="bias")
W = tf.Variable(tf.random_normal((1,1), seed=1), name="weight")

# Define equação linear do ajuste
y_pred = tf.matmul(X,W) + b

# Define função de custo
loss = tf.reduce_sum((y - y_pred)**2/n_samples)

# Define o otimizador como sendo o RMSprop com taxa de aprendizado
0,01
opt = tf.train.RMSPropOptimizer(0.01)

# Define a variável e a operação que o otimizador deve realizar
opt_operation = opt.minimize(loss)

# Imprime conteúdo das variáveis
print('b=', b)
print('W=', W)
print('y_pred=', y_pred)
print('Custo=', loss)
```

```
b= <tf.Variable 'bias_1:0' shape=(1,) dtype=float32_ref>
W= <tf.Variable 'weights_1:0' shape=(1, 1) dtype=float32_ref>
y_pred= Tensor("add_1:0", shape=(1000, 1), dtype=float32)
Custo= Tensor("Sum_1:0", shape=(), dtype=float32)
```

  - O viés `b` é inicializado com zero e o peso `W` com um número aleatório.
  - A função de custo utilizada é o erro quadrático médio.

- Observe que a impressão das variáveis nesse momento fornece somente o nome, a sua dimensão e o tipo de dado.

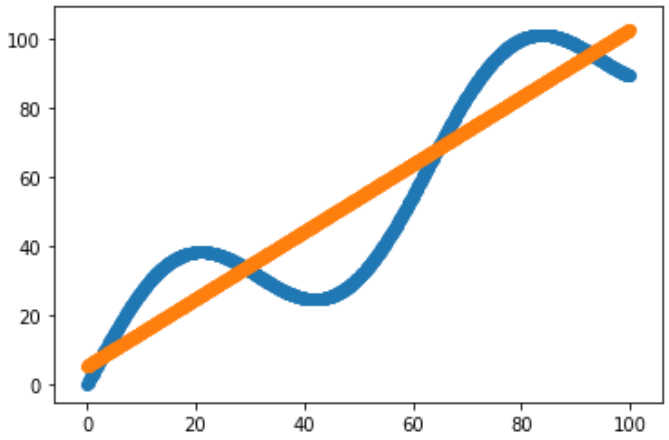
➢ No quadro a seguir é aberta e executada a sessão de otimização e cálculo dos parâmetros `W` e `b` para ajustar os dados com uma reta.

```
# Abre uma Sessão iterativa
sess = tf.InteractiveSession()

# Inicializa variables
sess.run(tf.global_variables_initializer())

# Imprime valores iniciais de W e b
print('W inicial=', W.eval(),'-','b inicial=', b.eval())

# Aplica processo de otimização por 1000 iterações
for _ in range(10000):
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X:
                            X_data, y: y_data})

# Imprime resultados
print('Valor do custo final=',loss_val)
print('W=', W.eval(),'-', 'b=', b.eval())

# Salva variáveis para serem usadas fora da sessão do TensorFlow
Ws = W.eval()
bs = b.eval()

# Fecha sessão
sess.close()
```

```
W inicial= [[-0.8113182]]; b inicial= [0.]
Valor do custo final= 176.48505
W= [[0.9733888]]; b= [5.2640986]
```

➢ No quadro a seguir é calculada a reta de ajuste aos dados e feito um gráfico das duas curvas para visualização.

```
# Calcula resultado do ajuste
y_prev = np.matmul(X_data,Ws) + bs

# Realiza gráfico dos resultados
plt.scatter(X_data, y_data)
plt.scatter(X_data, y_prev)
plt.show()
```



6. Execução do TensorFlow no modo “eager”

- O modo de execução eager (ansioso) do TensorFlow consiste de um ambiente de programação que calcula imediatamente as operações sem construir um gráfico computacional.
- No modo “eager” as operações retornam valores no lugar de um gráfico que é executado posteriormente.
- Esse modo de operação torna mais fácil depurar e testar programas e, assim, utilizar o TensorFlow.
- No quadro abaixo é mostrado como iniciar o modo “eager”.

```
import tensorflow as tf

# Habilitação do modo eager
tf.enable_eager_execution()

# Verificação se modo eager está ativo
tf.executing_eargerly()
```

```
True
2.2.4-tf
```

- Observa-se que esses comandos devem ser executados somente uma única vez no início do programa.
- Após habilitar o modo “eager” as operações realizadas com o TensorFlow retornam resultados imediatamente, como no exemplo mostrado no quadro a seguir.

```
X = tf.ones((2,2))
y = tf.matmul(x,x)
print(y)
print("hello,", format(y))
```

```
tf.Tensor(
[[2. 2.]
```

```
 [2. 2.]], shape=(2, 2), dtype=float32)
hello, [[2. 2.]
 [2. 2.]]
```

- Note a diferença entre os dois comandos `print`. O primeiro imprime todas as informações sobre o tensor `y` e o segundo somente o seu valor.

➢ O modo "eager" suporta a maioria das operações realizadas em GPU.

➢ Algumas operações que exigem muita memória podem ter problemas no modo "eager".

➢ Mais informações sobre o modo de execução "eager" pode ser obtido no manual do TensorFlow (https://www.tensorflow.org/guide/eager).

7. TensorBoard

➢ O TensorBoard é uma ferramenta que fornece meios para visualizar parâmetros e variáveis de um modelo do TensorFlow. Algumas funções do TensorBoard são:

- Rastrear os valores da função de custo e das métricas, visualizar o Gráfico Computacional do modelo, visualizar os parâmetros do modelo etc.
- Projetar tensores multidimensionais em um espaço de dimensão menor.

➢ Nessa seção são apresentados os comandos básicos para usar o TensorBoard. Mais detalhes de uso do TensorBoard podem ser obtidos em: https://www.tensorflow.org/tensorboard.

➢ Para usar o TensorBoard o seguinte comando deve ser incluído no início do seu programa.

```
%load_ext tensorboard
```

➢ A seguir importa-se o TensorFlow e a biblioteca datetime.

```
import tensorflow as tf
import datetime
```

- A biblioteca datetime serve para definir data e hora.

➢ O TensorBoard gera uma quantidade muito grande de arquivos e ocupa muita memória de disco. Assim, se você estiver desenvolvendo um modelo de rede neural é interessante antes de iniciar uma nova sessão do TensorBoard apagar os arquivos gerados em sessões anteriores. Para isso, deve-se usar os comandos no quadro a seguir.

```
import os,stat
import shutil

def remove_readonly(func, path,_):
    "Clear the readonly bit and reattempt the removal"
    os.chmod(path, stat.S_IWRITE)
    func(path)

shutil.rmtree("logs_date", onerror=remove_readonly)
```

- Nesse exemplo, os arquivos gerados pelo TensorBoard estão no diretório `logs_date`. Para obter o nome correto do diretório no seu Google Drive clicar com o botão direito do mouse no diretório e selecionar a opção `Copy path`.
- Obviamente, se o diretório `logs_date` não existir, esses comandos geram uma mensagem de erro, que nesse caso não tem nenhum problema.

➢ Nesse exemplo, vamos usar o conjunto de dados de dígitos numéricos da MNIST (https://en.wikipedia.org/wiki/MNIST_database) disponibilizado pelo Keras.

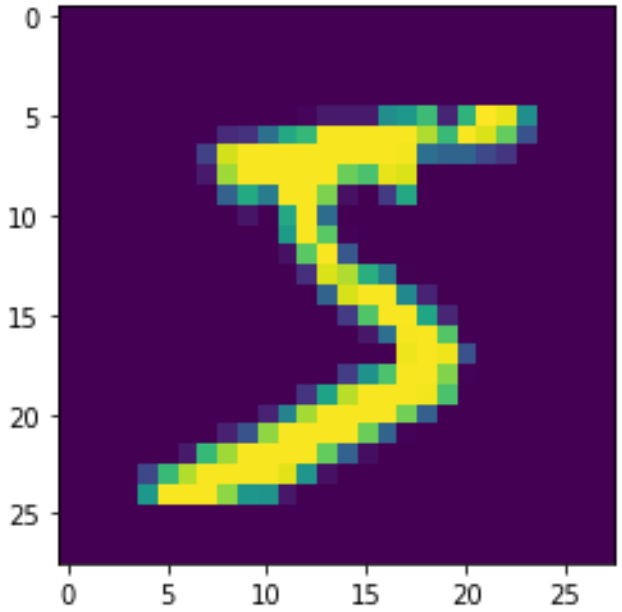
➢ As características desse conjunto de dados são as seguintes:

- Os dados de entrada consistem de imagens em tons de cinza de números, variando entre 0 a 9, escritos a mão;
- A dimensão das imagens é 28 por 28 pixels;
- A saída de cada exemplo consiste no número que representa a imagem;
- O conjunto de treinamento possui 60.000 exemplos e o conjunto de teste 10.000 exemplos.

➢ Os comandos do quadro abaixo carregam o conjunto de dados, normalizam os dados de entrada, apresentam a dimensão desses dados e mostra a imagem do dígito do primeiro exemplo de treinamento.

```
import matplotlib as plt

mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train/255.0, x_test/255.0

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)

index = 0
plt.imshow(x_train[index])
print("número = " + str(y_train[index]))
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
número = 5
```



- Como, queremos classificar 10 dígitos temos um problema de classificação multiclasse com 10 classes.

- A próxima etapa é criar uma RNA para aprender a classificar as imagens das 10 classes.

```
from tensorflow.keras import models
from tensorflow.keras import layers

def create_model():
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(28,28)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(10, activation='softmax'))
    return model

model = create_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

- É esperado que você consiga entender os comandos desse quadro.

➢ O treinamento da rede para usar o TensorBoard deve ser realizado com algumas variações:

- No método `fit` do Keras deve-se adicionar um callback para garantir que as informações geradas ao longo do treinamento da rede sejam armazenadas para serem usadas pelo TensorBoard. No caso, o callback usado é o `tf.keras.callbacks.TensorBoard`, que é específico para o TensorBoard.
- Ativar o cálculo do histograma para cada época do treinamento. Em um callback quando se usa a opção `histogram_freq=1`, o Keras calcula os histogramas das ativações dos neurônios e dos parâmetros das camadas da rede. Se usar `histogram_freq=0`, esses histogramas não são calculados. Mais informações sobre as opções de callbacks podem ser vistas em https://www.tensorflow.org/api_docs/python/tf/keras/callbacks.
- Guardar as informações em um subdiretório com data e hora para permitir identificar e selecionar as várias execuções.

```
#Compilação da rede usando o método de otimização ADAM
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#Cria diretório logs para guardar as informações do treinamento
logdir="logs" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

#Cria o callback para gerar e armazenar as informações do treinamento
tensorboard_callback =
                    tf.keras.callbacks.TensorBoard(log_dir=logdir,
                    histogram_freq=1)

#Treinamento da rede
model.fit(x=x_train, y=y_train, epochs=5, validation_data=(x_test,
          y_test), callbacks=[tensorboard_callback])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 7s 110us/sample - loss:
0.3699 - acc: 0.8916 - val_loss: 0.1771 - val_acc: 0.9478
Epoch 2/5
60000/60000 [==============================] - 6s 99us/sample - loss:
0.1927 - acc: 0.9430 - val_loss: 0.1235 - val_acc: 0.9646
Epoch 3/5
60000/60000 [==============================] - 6s 100us/sample - loss:
0.1556 - acc: 0.9529 - val_loss: 0.1161 - val_acc: 0.9650
Epoch 4/5
60000/60000 [==============================] - 6s 101us/sample - loss:
0.1309 - acc: 0.9613 - val_loss: 0.0960 - val_acc: 0.9708
Epoch 5/5
60000/60000 [==============================] - 6s 99us/sample - loss:
0.1185 - acc: 0.9642 - val_loss: 0.0944 - val_acc: 0.9710
<tensorflow.python.keras.callbacks.History at 0x7fd926a35320>
```

- Observe que a função de custo utilizada é a `sparse_categorical_crossentropy`, que é diferente da `categorical_crossentropy` que foi usada para problemas de classificação multiclasse na Aula 12. A função `sparse_categorical_crossentropy` não necessita que as saídas sejam transformadas pela função `one_hot_encoding`, pois isso é feito dentro da própria função.

➢ Para verificar se as informações do treinamento foram salvas podemos listar o conteúdo do diretório gerado no treinamento.

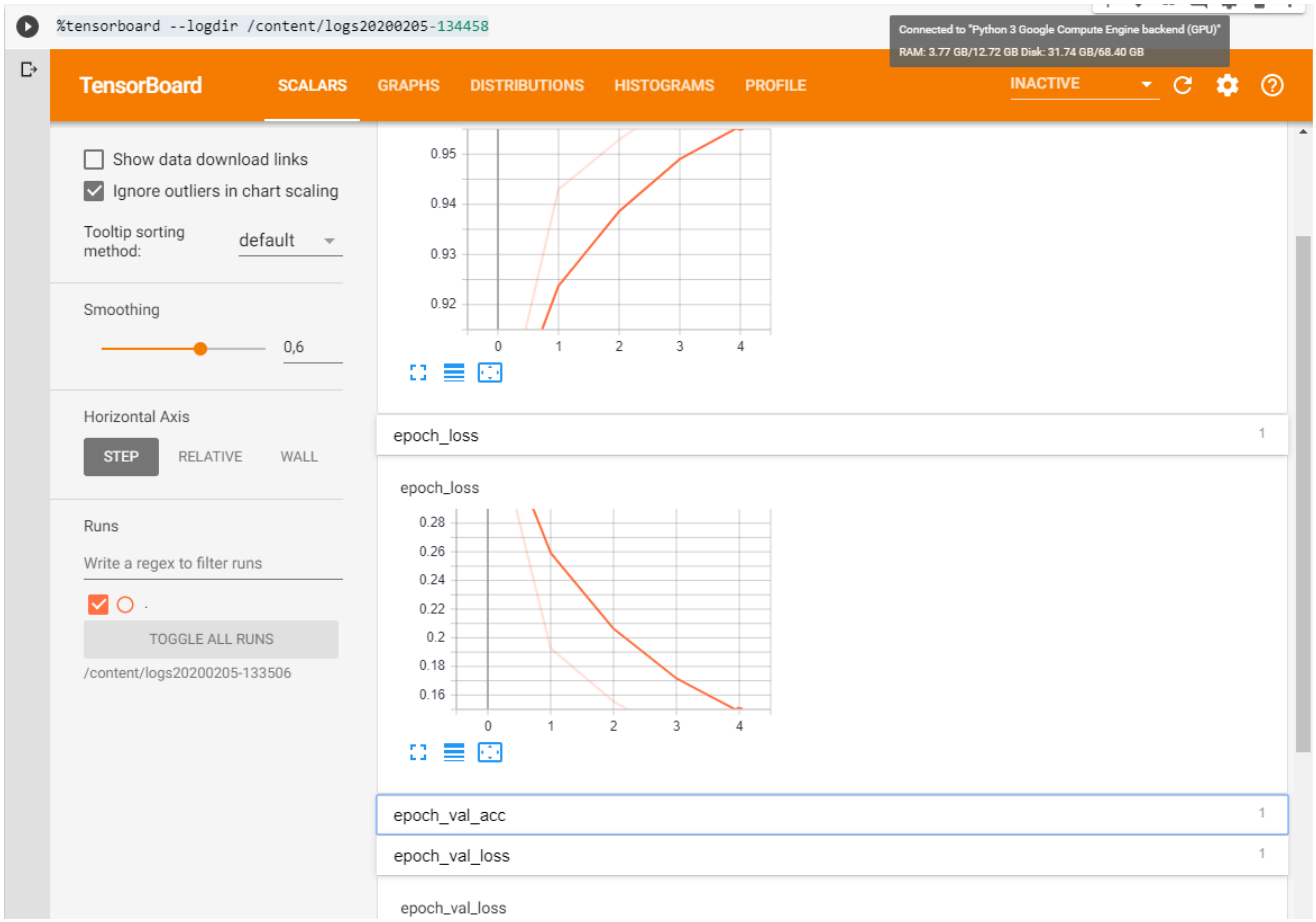
```
!ls logs_date
```

```
events.out.tfevents.1580910298.e164e3e4f751        plugins
events.out.tfevents.1580910298.e164e3e4f751.profile-empty
```

- `logs_date` é o nome do diretório no seu Google Drive. Para obter o nome correto clicar com o botão direito do mouse no diretório e selecionar a opção `Copy path`.

➢ Iniciar o TensorBoard usando o comando do quadro a seguir.

```
%tensorboard --logdir logs_date
```



➢ Uma breve visão geral das opções mostradas na barra de navegação superior laranja:

- A opção `SCALARS` mostra os valores da função de custo e das métricas para cada época de treinamento. Você pode usá-lo para também rastrear a velocidade do treinamento, a taxa de aprendizado e outros valores escalares.
- A opção `GRAPHS` ajuda a visualizar seu modelo. Nesse caso, um gráfico com as camadas da rede é mostrado, o que pode ajudar a garantir que a rede não tenha erros de conexões.
- As opções `DISTRIBUTIONS` e `HISTOGRAMS` mostram a variação de um tensor ao longo do tempo. Isso pode ser útil para visualizar pesos e vieses e verificar se eles estão mudando da maneira esperada.
- Outras opções adicionais do TensorBoard são ativados automaticamente quando se usa outros tipos de dados. Por exemplo, é possível guardar imagens. Pode-se ver quais outras opções estão disponíveis no TensorBoard clicando no menu suspenso `INATIVO` no canto superior direito.
- Mais informações das opções do TensorBoard pode ser vistas em https://www.tensorflow.org/tensorboard.