TensorFlow

Alex Fernandes Mansano

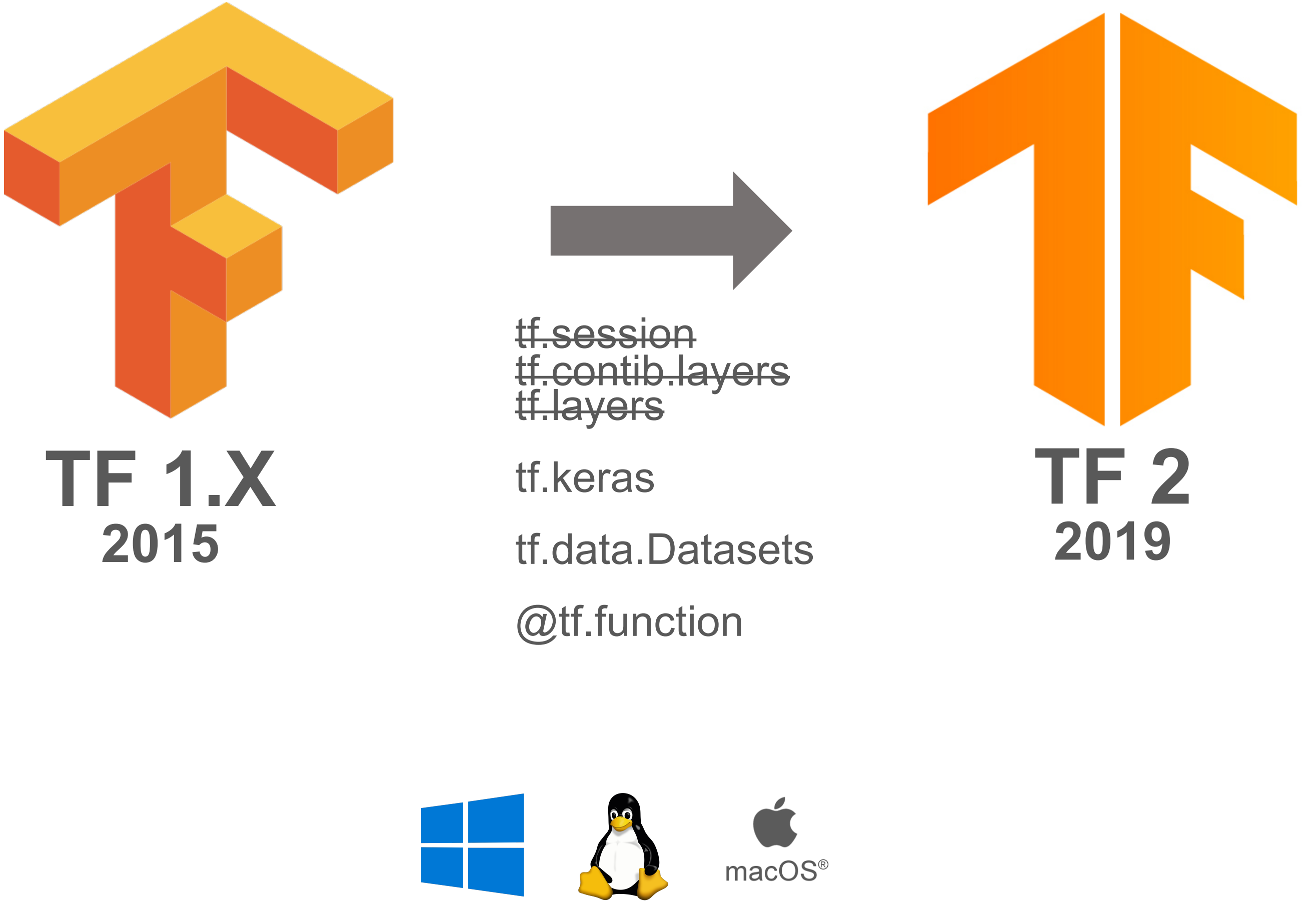
Lead Data Scientist – Itaú Unibanco

MSc. Aprendizado de Máquina – Ufscar

BSc. Eng. Sistemas de Informação – Institut National Polytechinique de Grenoble

BSc. Sistemas de Informação – Unesp

TensorFlow



TF 1.X
2015

~~tf.session~~
~~tf.contib.layers~~
~~tf.layers~~

tf.keras

tf.data.Datasets

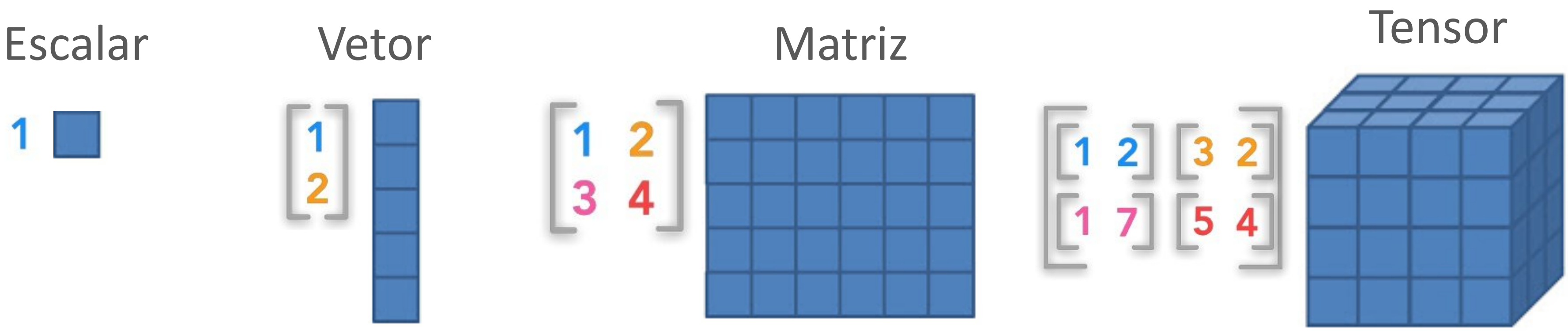
@tf.function

TF 2
2019

macOS®

Tensor

Um tensor pode ser visto como a generalização de um vetor ou matriz em *n* dimensões



0 dimensão

1 dimensão

2 dimensões

3 ou mais dimensões

TensorFlow

Constantes

```
n = np.array([[1., 2., 3., 4.], [5., 6., 7., 8.]])
print(n)
[[1. 2. 3. 4.]
[5. 6. 7. 8.]]
```

```
t = tf.constant([[1., 2., 3., 4.], [5., 6., 7., 8.]])
print(t)
tf.Tensor(
[[1. 2. 3. 4.]
 [5. 6. 7. 8.]], shape=(2, 4), dtype=float32)
```

Dimensões e tipos

```
print(’shape:’, n.shape)
print(’dtype:’, n.dtype)

shape: (2,4)
dtype: float64
```

```
print(’shape:’, t.shape)
print(’dtype:’, t.dtype)

shape: (2,4)
dtype: <dtype: 'float32'>
```

TensorFlow

Operações

```
tf.square(t)
np.square(n)

[[ 1., 4., 9., 16.],
[25., 36., 49., 64.]]
```

```
np.reshape(n, (2,2,2))
tf.reshape(t, (2,2,2))

[[[1., 2.],
 [3., 4.]],

[[5., 6.],
[7., 8.]]]
```

```
np.stack([n, n])
Tf.stack([t, t])

[[[1., 2., 3., 4.],
[5., 6., 7., 8.]],

[[1., 2., 3., 4.],
[5., 6., 7., 8.]]]
```

TensorFlow

Operações

```
n.T
tf.transpose(t)

[[1., 5.],
 [2., 6.],
[3., 7.],
[4., 8.]]
```

```
np.sum(n, axis=1)
tf.reduce_sum(t, axis=1)

[6., 15.]
```

```
n @ n.T
t @ t.transpose(t)

[[ 30., 70.],
[ 70., 174.]]
```

```
np.mean(n, axis=0)
tf.reduce_mean(t, axis=0)

[3., 4., 5., 6.]
```

TensorFlow

Operações

```
t + 10
tf.add(t, 10., name=’adicao’)


t - 10
tf.subtract(t, 10., name=’subtracao’)

t / 10
tf.divide(t, 10., name=’divisao’)

t * 10
tf.muliply(t, 10., name=’multiplicacao’)
```

TensorFlow

Conversões

```
a = tf.constant(6.)
<tf.Tensor: shape=(), dtype=float32, numpy=6.0>

b = tf.constant(4)
<tf.Tensor: shape=(), dtype=int32, numpy=4>

tf.cast(b, dtype=tf.float32)
<tf.Tensor: shape=(), dtype=float32, numpy=4.0>


a + tf.cast(b, dtype=tf.float32)
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>
```

TensorFlow

Variáveis

```
t[1, 2].assign(30)


v = tf.Variable([[1., 2., 3.], [4., 5., 6.]])
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 2., 3.], [4., 5., 6.]], dtype=float32)>

v[1,2].assign(30)
[[ 1., 2., 3.],
 [ 4., 5., 30.]]
```

TensorFlow

Autodiff

```python
def f(w1, w2):
    return 3 * w1 ** 2 + 2 * w1 * w2




w1, w2 = tf.Variable(5.), tf.Variable(3.)
with tf.GradientTape() as tape:
     z = f(w1, w2)

gradients = tape.gradient(z, [w1, w2])
```

```
[<tf.Tensor: shape=(), dtype=float32, numpy=36.0>,
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>]
```

$$3 \times w1^2 + 2 \times w1 \times w2$$

TensorFlow

Autodiff

```python
def f(w1, w2):
   return 3 * w1 ** 2 + 2 * w1 * w2
```

```python
w1, w2 = tf.Variable(5.), tf.Variable(3.)
with tf.GradientTape() as tape:
     z = f(w1, w2)

gradients = tape.gradient(z, [w1, w2])
```

```
[<tf.Tensor: shape=(), dtype=float32, numpy=36.0>,
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>]
```

e se w1 e w2 fossem constantes?

$3 \times w1^2 + 2 \times w1 \times w2$

TensorFlow

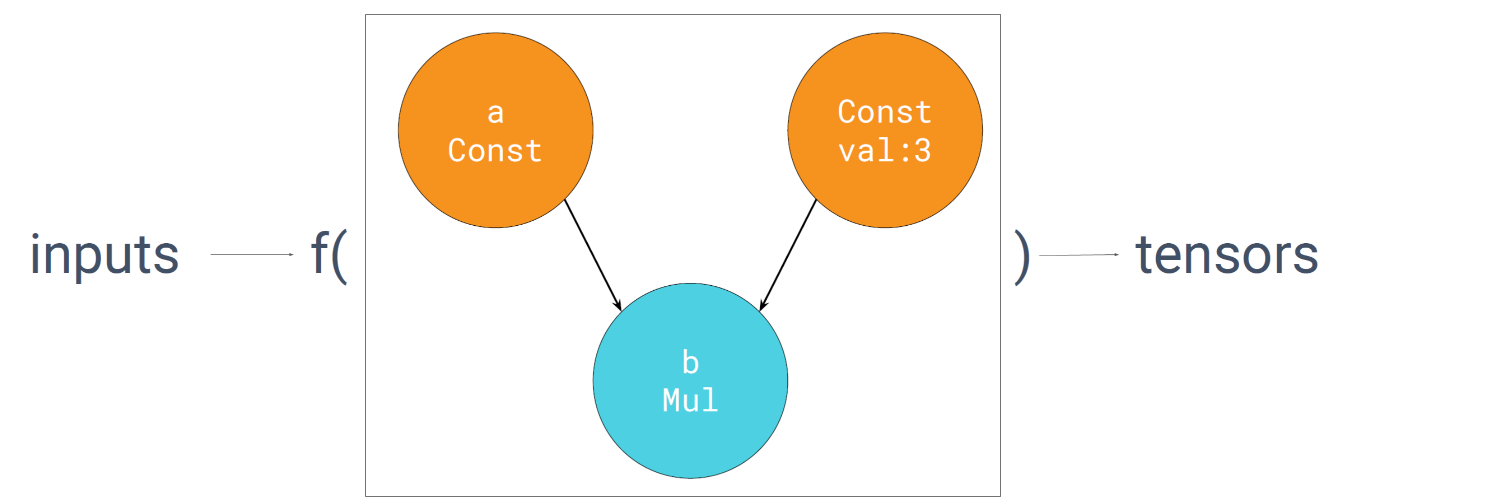
TF functions

```
def cubo(x):
   return x**3

cubo(np.array(2, 4))
[8., 16.]
```

Recebe np.array, retorna np.array

TF functions

```
@tf.function
```



TensorFlow

TF functions

```
@tf.function
def cubo(x):
   return x**3

cubo(np.array(2, 4))
<tf.Tensor: shape=(2,), dtype=int64,
numpy=array([27, 64])>
```

=

```
def cubo(x):
   return tf.power(x,3)

cubo(np.array(2, 4))
<tf.Tensor: shape=(2,), dtype=int64,
numpy=array([27, 64])>
```

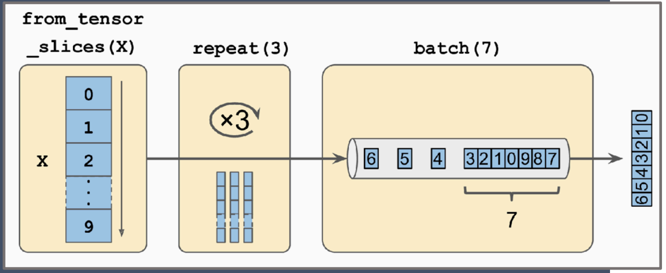
TensorFlow

TF data API

Na maior parte dos projetos de deep learning, o dataset não cabe interinamente na RAM

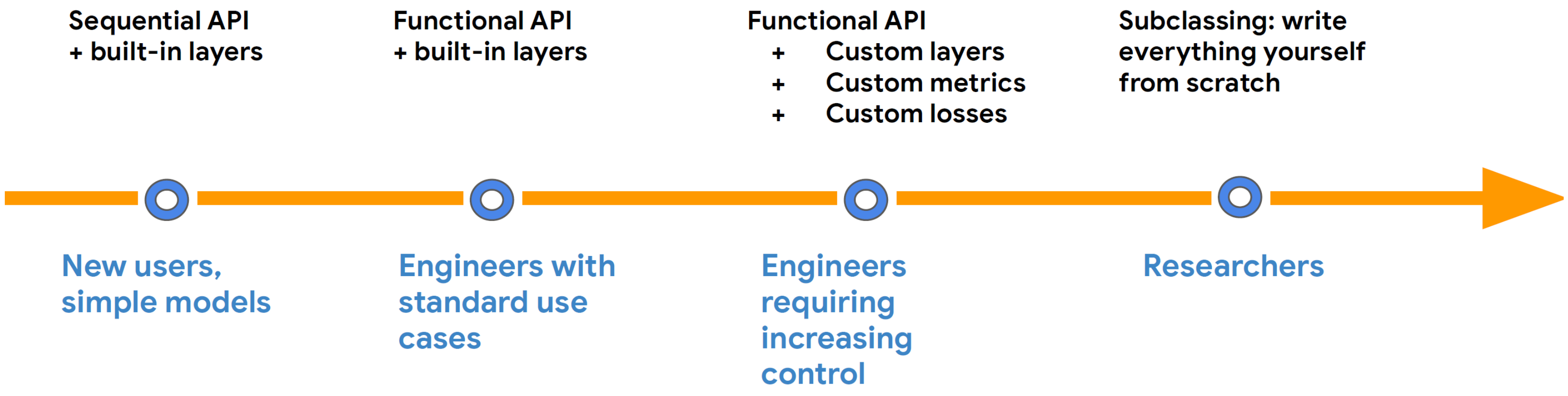
Tensorflow disponibiliza o data API para realizar pré-processamento, carregamento em batch, multithreading com o tf.keras

```
X = tf.range(10)
dataset = tf.data.Dataset.from_tensor_slices(X)
<TensorSliceDataset shapes: (), types: tf.int32>

ndataset = dataset.repeat(3).batch(7)
tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int32)
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int32)
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int32)
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int32)

ndataset.map()
```



Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity



Sequential API
+ built-in layers

New users,
simple models

Functional API
+ built-in layers

Engineers with
standard use
cases

Functional API
- + Custom layers
- + Custom metrics
- + Custom losses

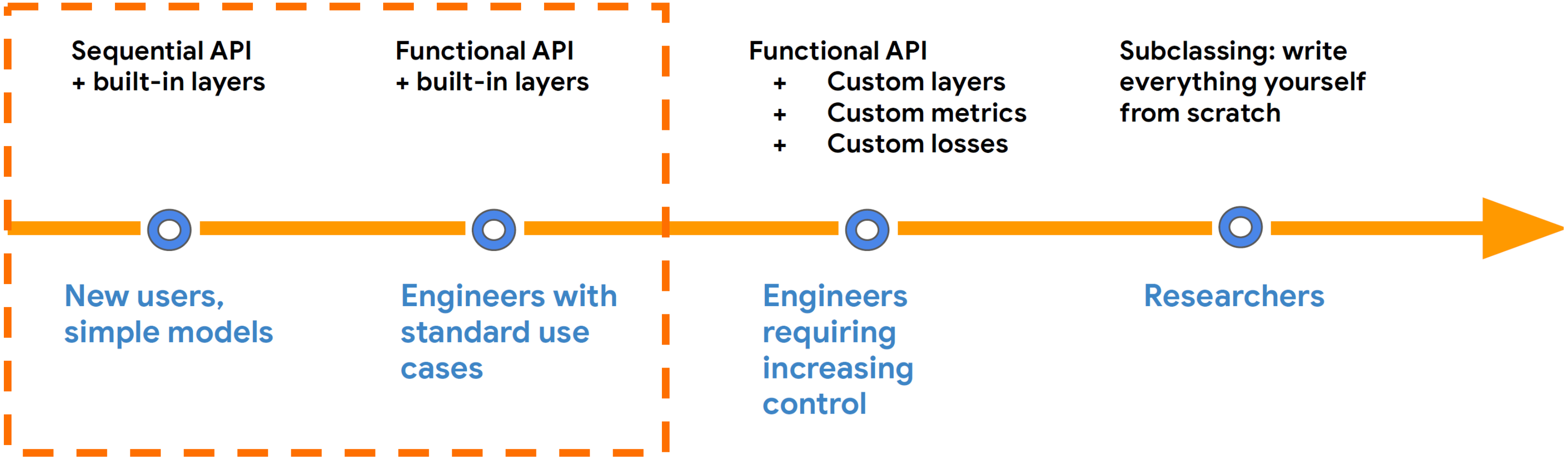
Engineers
requiring
increasing
control

Subclassing: write
everything yourself
from scratch

Researchers

Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity



Sequential API
+ built-in layers

New users, simple models

Functional API
+ built-in layers

Engineers with standard use cases

Functional API
+ Custom layers
+ Custom metrics
+ Custom losses

Engineers requiring increasing control

Subclassing: write everything yourself from scratch

Researchers

TensorFlow

Sequential API

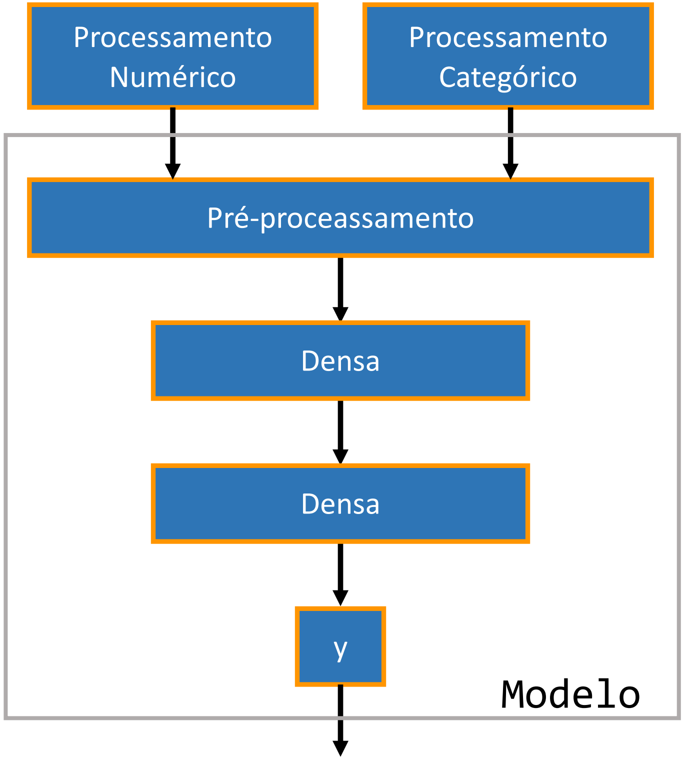
API simples para criação rápida de redes neurais ”clássicas”

```
# Define um modelo Sequential com 3 camadas
model = tf.keras.Sequential([
       layers.Dense(2, activation="relu", name="layer1"),
       layers.Dense(3, activation="relu", name="layer2"),
       layers.Dense(4, name="layer3"),
])


# Executa o modelo em em exemplo de teste
x = tf.ones((3, 3))
y = model(x)
```

Sequential API

notebook: `NNRegression.ipynb`

Boston House Price



Functional API

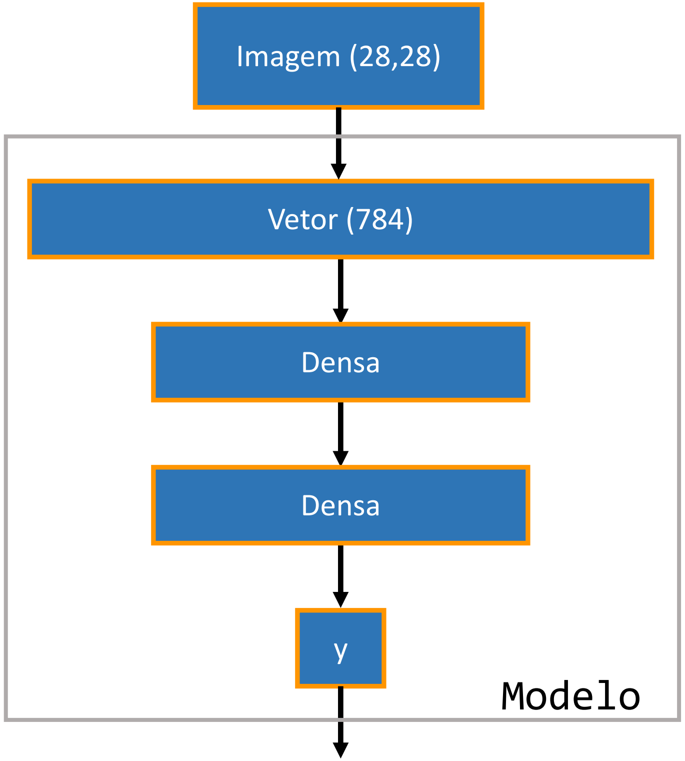
Permite maior maleabilidade do que o Sequential.
Topologias não lineares, pesos compartilhados...

```
#define as dimensões do vetor de entrada
inputs = tf.keras.Input(shape=(3,))
x = tf.keras.layers.Dense(128, activation=tf.nn.relu)(inputs)
x = tf.keras.layers.Dense(128, activation=tf.nn.relu)(x)
outputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)

#define o modelo com a camada de entrada e saída
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

Functional API

notebook: `NNSimpleClassification.ipynb`

Fashion Mnist



Obrigado!