Bootcamp: Engenheiro(a) de Machine Learning

Trabalho Prático

Módulo 4	Metodologias de Aprendizado

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Neste trabalho prático será utilizado o ambiente de desenvolvimento do Google Colab. Para acessar o ambiente, basta ter uma conta do Google ativa e acessar o Google Drive.

2. Executar a prática abaixo.

Objetivos

O objetivo deste exercício é classificar imagens através do uso de rede neural. Vamos treinar um modelo de rede neural para classificação de imagens de roupas. Teremos uma visão geral do uso do TensorFlow.

https://www.tensorflow.org/tutorials/keras/classification?hl=pt-br

Enunciado

Vamos usar o Keras para construir e treinar modelos no TensorFlow.

```
# TensorFlow e tf.keras
import tensorflow as tf
from tensorflow import keras

# Librariesauxiliares
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

Versão 2.2.0.

Importar a base de dados Fashion MNIST

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Carregando a base de dados que retorna quatro NumPyarrays:

- Os arraystrain_images e train_labels são o conjunto de treinamento, os dados do modelo usados para aprender.
- O modelo é testado com o conjunto de teste, os arraystest_images e test_labels.

As imagens são arraysNumPy de 28x28, com os valores de pixels entre 0 to 255. Oslabels (alvo da classificação) são um array de inteiros, no intervalo de 0 a 9.

Esse intervalo corresponde com a classe de roupa que cada imagem representa.

Label	Classe
0	Camisetas/Top (T-shirt/top)
1	Calça (Trouser)
2	Suéter (Pullover)
3	Vestidos (Dress)
4	Casaco (Coat)
5	Sandálias (Sandal)
6	Camisas (Shirt)
7	Tênis (Sneaker)
8	Bolsa (Bag)
9	Botas (Ankle boot)

Cada imagem é mapeada com um só label, já que os *nomes das classes* não são incluídos na base de dados. Armazene os dados aqui para usá-los mais tarde quando formos plotar as imagens.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Explore os dados

Vamos explorar o formato da base de dados antes de treinar o modelo. O próximo comando mostra que existem 60.000 imagens no conjunto de treinamento, e cada imagem é representada em 28 x 28 pixels.

```
train_images.shape
```

(60000, 28, 28)

Do mesmo modo, existem 60.000 labels no conjunto de treinamento.

```
len(train_labels)
```

Cada label é um inteiro entre 0 e 9.

```
train_labels
```

array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)

Existem 10000 imagens no conjunto de teste. Novamente, cada imagem é representada por 28 x 28 pixels.

```
test_images.shape
```

(10000, 28, 28)

E um conjunto de teste contendo 10000 labels das imagens.

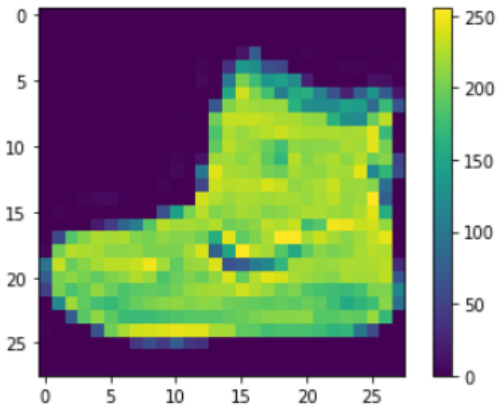
```
len(test_labels)
```

10000

Pré-processe os dados

Os dados precisam ser pré-processados antes de treinar a rede. Se você inspecionar a primeira imagem do conjunto de treinamento, você verá que os valores dos pixels estão entre 0 e 255.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



Escalaremos esses valores no intervalo de 0 e 1 antes de alimentar o modelo da rede neural. Para fazer isso, dividiremos os valores por 255. É importante que o *conjunto de treinamento* e o *conjunto de teste* possam ser pré-processados do mesmo modo.

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

Para verificar que os dados estão no formato correto e que estamos prontos para construir e treinar a rede, vamos mostrar as primeiras 25 imagens do *conjunto de treinamento* e mostrar o nome das classes de cada imagem abaixo.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Construindo o modelo

Construir a rede neural requer configurar as camadas do modelo e, depois, compilar o modelo.

Montar as camadas

O principal bloco de construção da rede neural é a camada (*layer*). As camadas (*layers*) extraem representações dos dados inseridos na rede. Com sorte, essas representações são significativas para o problema.

Muito do *deeplearning* consiste em encadear simples camadas. Muitas camadas, como tf.keras.layers.Dense, têm parâmetros que são aprendidos durante o treinamento.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

A primeira camada da rede, tf.keras.layers.Flatten, transforma o formato da imagem de um array de imagens de duas dimensões (of 28 by 28 pixels) para um array de uma dimensão (de 28 * 28 = 784 pixels). Pense nessa camada como camadas não empilhadas de pixels de uma imagem e os enfileire. Essa camada não tem parâmetros para aprender; ela só reformata os dados.

Depois dos pixels serem achatados, a rede consiste de uma sequência de duas camadas tf.keras.layers.Dense. Essas são camadas neurais *denselyconnected*, ou *fullyconnected*. A primeira camada Dense tem 128 nós (ou neurônios). A segunda (e última) camada é uma *softmax* de 10 nós, que retorna um array de 10 probabilidades, cuja soma resulta em 1. Cada nó contém um valor que indica a probabilidade de que aquela imagem pertença a uma das 10 classes.

Compile o modelo

Antes do modelo estar pronto para o treinamento, são necessárias algumas configurações. Essas configurações serão adicionadas no passo de *compilação*:

- *Função Loss:* essa função mede quão preciso o modelo é durante o treinamento. Queremos minimizar a função para *guiar* o modelo para a direção certa.
- *Optimizer*: isso é como o modelo se atualiza com base no dado que ele vê e sua função *loss*.
- *Métricas:* usadas para monitorar os passos de treinamento e teste. O exemplo abaixo usa a *acurácia*, a fração das imagens que foram classificadas corretamente.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Treine o modelo

Treinar a rede neural requer os seguintes passos:

- Alimente o modelo com os dados de treinamento. Neste exemplo, os dados de treinamento são os arrays train_images e train_labels.
- O modelo aprende como associar as imagens às labels.
- Perguntamos ao modelo para fazer previsões sobre o conjunto de teste — nesse exemplo, o array test_images. Verificamos se as previsões combinaram com as labels do array test_labels.

Para começar a treinar, chame o método model.fit — assim chamado porque ele "encaixa" o modelo no conjunto de treinamento:

```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5005 - accuracy: 0.8259
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3766 - accuracy: 0.8646
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3411 - accuracy: 0.8753
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3132 - accuracy: 0.8852
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2956 - accuracy: 0.8914
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2810 - accuracy: 0.8959
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2687 - accuracy: 0.9007
Epoch 8/10
```

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2558 - accuracy: 0.9053
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2468 - accuracy: 0.9080
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2375 - accuracy: 0.9102
<tensorflow.python.keras.callbacks.History at 0x7fe168897198>
```

À medida que o modelo treina, as métricas loss e acurácia são mostradas. O modelo atinge uma acurácia de 0.88 (ou 88%) com o conjunto de treinamento.

Avalie a acurácia

Depois, compare como o modelo performou com o conjunto de teste.

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 0.3418 - accuracy: 0.8811

Test accuracy: 0.8810999989509583
```

A acurácia com o conjunto de teste é um pouco menor do que a acurácia de treinamento. Essa diferença entre as duas acurácias representa um *overfitting*. Overfitting é um modelo de aprendizado de máquina que performou de maneira pior em um conjunto de entradas novas e não usadas anteriormente.

Faça predições

Com o modelo treinado, o usaremos para predições de algumas imagens.

```
predictions = model.predict(test_images)
```

Aqui, o modelo previu a *label* de cada imagem no conjunto de treinamento. Vamos olhar na primeira predição.

```
predictions[0]
```

```
array([3.1997556e-07, 5.0947659e-07, 1.8399154e-08, 2.9958559e-07,
```

```
8.0525624e-08, 1.7922817e-03, 2.9665678e-06, 6.6630309e-03,
1.1742719e-07, 9.9154037e-01], dtype=float32)
```

A predição é um array de 10 números. Eles representam uma *confiança* do modelo que a imagem corresponde a cada um dos diferentes artigos de roupa. Podemos ver que cada *label* tem um maior valor de confiança.

```
np.argmax(predictions[0])
```

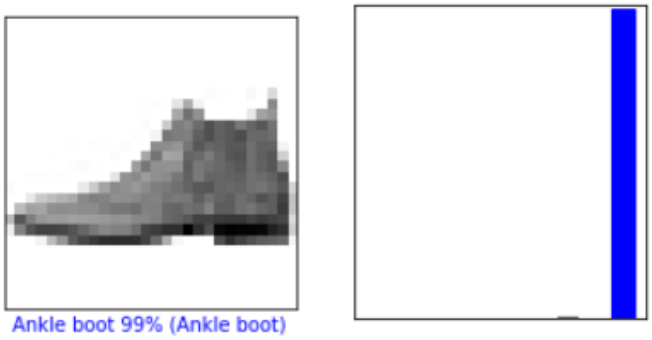
```
9
```

Podemos mostrar graficamente como se parece em um conjunto total de previsão de 10 classes.

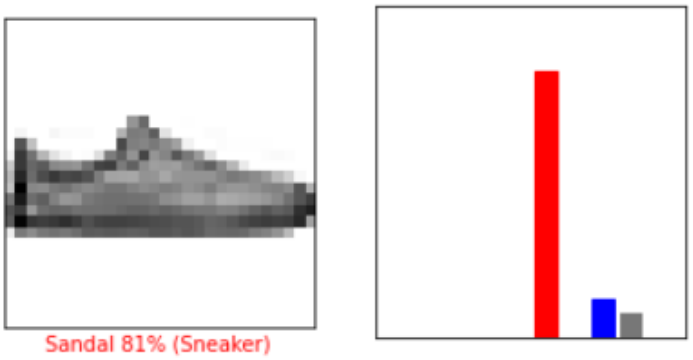
```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Vamos olhar a previsão imagem na posição 0, do array de predição.

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions,  test_labels)
plt.show()
```

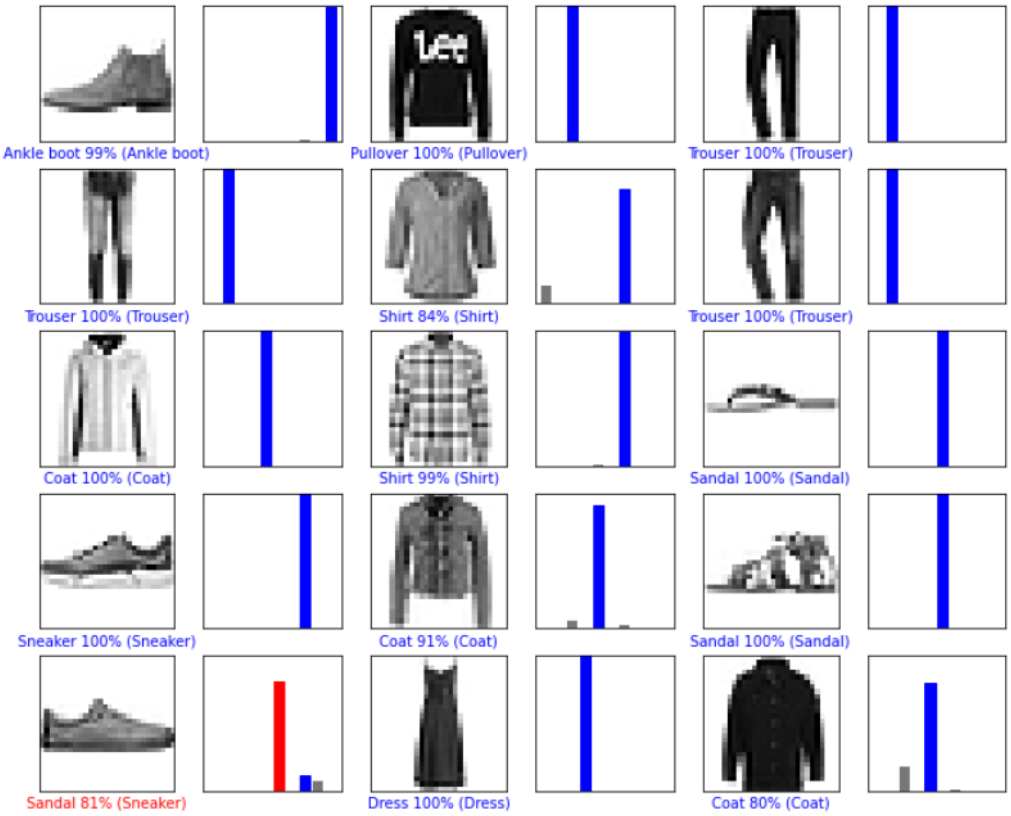



```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions,  test_labels)
plt.show()
```



Vamos plotar algumas das previsões do modelo. Labels preditas corretamente são azuis e as predições erradas são vermelhas. O número dá a porcentagem (de 100) das labels preditas. Note que o modelo pode errar.

```
# Plota o primeiro X test images, e as labels preditas, e as labels verdadeiras.
# Colore as predições corretas de azul e as incorretas de vermelho.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)
plt.show()
```



Finalmente, use o modelo treinado para fazer a predição de uma única imagem.

```
# Grab an image from the test dataset.
img = test_images[0]

print(img.shape)
```

(28, 28)

Modelos tf.keras são otimizados para fazer predições em *batch*, ou coleções, de exemplos de uma vez. Assim, mesmo que usemos uma única imagem, precisamos adicionar em uma lista.

```
# Adiciona a imagem em um batch que possui um só membro.
img = (np.expand_dims(img,0))

print(img.shape)
```

(1, 28, 28)

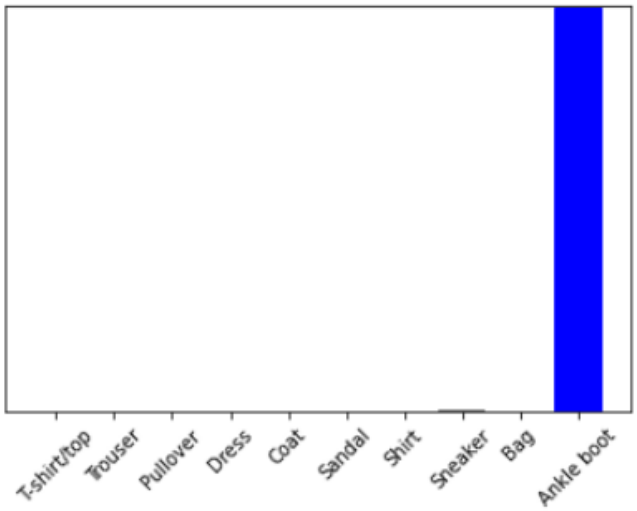
Agora faremos a predição da label correta para essa imagem.

```
predictions_single = model.predict(img)

print(predictions_single)
```

[[3.1997527e-07 5.0947659e-07 1.8399154e-08 2.9958559e-07 8.0525780e-08

1.7922842e-03 2.9665650e-06 6.6630342e-03 1.1742696e-07 9.9154037e-01]]

```
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```



model.predict retorna a lista de listas — uma lista para cada imagem em um *batch* de dados. Pegue a predição de nossa (única) imagem no *batch*.

```
np.argmax(predictions_single[0])
```

9

E, como antes, o modelo previu a label como 9.

Usamos a base de dados Fashion MNIST, que contém 70,000 imagens em tons de cinza em 10 categorias. As imagens mostram artigos individuais de roupas com baixa resolução (28 por 28 pixels).

Fashion MNIST tem como intenção substituir a clássica base de dados MNIST.

Usamos 60,000 imagens para treinar nossa rede e 10,000 imagens para avaliar quão precisamente nossa rede aprendeu a classificar as imagens. Se quiser, você pode acessar a Fashion MNIST diretamente do TensorFlow.