hy-23 Created using Colaboratory History

1 contributor

871 lines (871 sloc) | 27.7 KB

Open in Colab

Introduction

KerasTuner is a general-purpose hyperparameter tuning library. It has strong integration with Keras workflows, but it isn't limited to them: you could use it to tune scikit-learn models, or anything else. In this tutorial, you will see how to tune model architecture, training process, and data preprocessing steps with KerasTuner. Let's start from a simple example.

https://keras.io/guides/keras_tuner/getting_started/

In [1]:
```
!pip install keras-tuner -q
```

```
     |████████████████████████████████| 133 kB 8.3 MB/s
```

Import

In [2]:
```
import tensorflow as tf
from tensorflow import keras
from keras import layers

import keras_tuner as kt
import numpy as np


print(tf.__version__)
```

```
2.8.0
```

Functions

In [3]:
```
def build_model_units(hp):
    model = keras.Sequential()
    model.add(layers.Flatten())
    model.add(
        layers.Dense(
            # Define the hyperparameter.
            units=hp.Int("units", min_value=32, max_value=512, step=32),
            activation="relu",
        )
    )
    model.add(layers.Dense(10, activation="softmax"))
    model.compile(
        optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"],
    )
    return model
```

In [4]:
```
def build_model_units_activation_dropout_lr(hp):
    model = keras.Sequential()
    model.add(layers.Flatten())
    model.add(
        layers.Dense(
            # Tune number of units.
            units=hp.Int("units", min_value=32, max_value=512, step=32),
            # Tune the activation function to use.
            activation=hp.Choice("activation", ["relu", "tanh"]),
        )
    )
    # Tune whether to use dropout.
    if hp.Boolean("dropout"):
        model.add(layers.Dropout(rate=0.25))

    model.add(layers.Dense(10, activation="softmax"))

    # Define the optimizer learning rate as a hyperparameter.
    learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss="categorical_crossentropy",
```

```
        metrics=["accuracy"],
    )
    return model
```

In [5]:
```
hp = kt.HyperParameters() # Hyperparameter class.
print(hp.Int("units", min_value=32, max_value=512, step=32))
```

```
32
```

In [6]:
```
# Template for build and compile.

def call_existing_code(units, activation, dropout, lr):
    model = keras.Sequential()
    model.add(layers.Flatten())
    model.add(layers.Dense(units=units, activation=activation))
    if dropout:
        model.add(layers.Dropout(rate=0.25))
    model.add(layers.Dense(10, activation="softmax"))
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=lr),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model
```

In [7]:
```
def build_model(hp):
  units = hp.Int("units", min_value=32, max_value=512, step=32)
  activation = hp.Choice("activation", ['relu', 'tanh'])
  dropout = hp.Boolean('dropout')
  lr = hp.Float('lr', min_value=1e-4, max_value=1e-2, sampling='log')

  model = call_existing_code(units=units, activation=activation, dropout=dropout, lr=lr)
  return model
```

In [8]:
```
# check if the model builds.
hp = kt.HyperParameters()
build_model(hp)
```

Out[8]:
```
<keras.engine.sequential.Sequential at 0x7f961326f350>
```

In [9]:
```
def build_compile_model(hp):
    model = keras.Sequential()
    model.add(layers.Flatten())
    # Tune the number of layers.
    for i in range(hp.Int("num_layers", 1, 3)):
      print('number of layers')
      model.add(
      layers.Dense(
          # Tune number of units separately.
          units=hp.Int(f"units_{i}", min_value=32, max_value=512, step=32),
          activation=hp.Choice("activation", ["relu", "tanh"]),
          )
      )

    print()
    print('Dropout is {}'.format(hp.Boolean('dropout')))
    if hp.Boolean("dropout"):
        model.add(layers.Dropout(rate=0.25))
    model.add(layers.Dense(10, activation="softmax"))

    learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")
    print('learning rate is {}'.format(learning_rate))

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )


    return model

build_model(kt.HyperParameters())
```

Out[9]:
```
<keras.engine.sequential.Sequential at 0x7f9589392f10>
```

In [10]:
```
tuner = kt.RandomSearch(
```

```
    hypermodel=build_compile_model,
    objective="val_accuracy",
    max_trials=3,
    executions_per_trial=2,
    overwrite=True,
    directory="my_dir",
    project_name="helloworld",
)
```

```
number of layers

Dropout is False
learning rate is 0.0001
```

In [11]:

```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 5
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
dropout (Boolean)
{'default': False, 'conditions': []}
lr (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step': None, 'sampling':
'log'}
```

Load the dataset

In [12]:

```
(x, y), (x_test, y_test) = keras.datasets.mnist.load_data()

# In axis0, until last 10000 elements. And, all elements in all other axes.
x_train = x[:-10000]
y_train = y[:-10000]

# In axis0, last 10000 elements. And, all elements in all other axes.
x_val = x[-10000:]
y_val = y[-10000:]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [13]:

```
print('shape of x dataset {}'.format(x.shape))
print('shape of y dataset {}'.format(y.shape))

print()

print('shape of xtrain dataset {}'.format(x_train.shape))
print('shape of ytrain dataset {}'.format(y_train.shape))
```

```
shape of x dataset (60000, 28, 28)
shape of y dataset (60000,)

shape of xtrain dataset (50000, 28, 28)
shape of ytrain dataset (50000,)
```

In [14]:

```
# Add a new dimension at the end. Need?
de_x_train = np.expand_dims(x_train, axis=-1)
de_x_val   = np.expand_dims(x_val, axis=-1)
de_x_test  = np.expand_dims(x_test, axis=-1)

print('shape of dimension extended x_train {}'.format(de_x_train.shape))
print('shape of dimension extended x_val {}'.format(de_x_val.shape))
print('shape of dimension extended x_test {}'.format(de_x_test.shape))
```

```
shape of dimension extended x_train (50000, 28, 28, 1)
shape of dimension extended x_val (10000, 28, 28, 1)
shape of dimension extended x_test (10000, 28, 28, 1)
```

In [15]:

```
# Scale the value between 0 and 1.
de_x_train = de_x_train.astype('float32') / 255.0
de_x_val = de_x_val.astype('float32') / 255.0
de_x_test = de_x_test.astype('float32') / 255.0
```

In [16]:

In [16]:
```
num_classes = 10
ca_y_train = keras.utils.to_categorical(y_train, num_classes)
ca_y_val = keras.utils.to_categorical(y_val, num_classes)
ca_y_test = keras.utils.to_categorical(y_test, num_classes)
```

In [17]:
```
print('shape of y_train is {}'.format(y_train.shape))
print('shape of ca_y_train is {}'.format(ca_y_train.shape))

print()

print('example of {}: {}'.format(y_train[5], ca_y_train[5]))
print('example of {}: {}'.format(y_train[15], ca_y_train[15]))
print('example of {}: {}'.format(y_train[25], ca_y_train[25]))
print('example of {}: {}'.format(y_train[35], ca_y_train[35]))
```

```
shape of y_train is (50000,)
shape of ca_y_train is (50000, 10)

example of 2: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
example of 7: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
example of 2: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
example of 5: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In [18]:
```
a = np.array([[1,2,3], [11,12,13], [21,22,23], [31,32,33]])

print(a[:2]) # print from [beginning, 2]
print()
print(a[2:]) # print from (2, everything]
```

```
[[ 1  2  3]
 [11 12 13]]

[[21 22 23]
 [31 32 33]]
```

In [19]:
```
tuner.search(de_x_train, ca_y_train, epochs=2, validation_data=(de_x_val, ca_y_val))
```

```
Trial 3 Complete [00h 00m 42s]
val_accuracy: 0.9628500044345856

Best val_accuracy So Far: 0.9671000242233276
Total elapsed time: 00h 01m 30s
INFO:tensorflow:Oracle triggered exit
```

Get the best models

In [20]:
```
models = tuner.get_best_models(num_models=2)
best_model = models[0]

# Build the model
best_model.build(input_shape=(None, 28, 28))
best_model.summary()
```

```
number of layers

Dropout is False
learning rate is 0.003449485159787074
number of layers
number of layers

Dropout is False
learning rate is 0.0003271739205768162
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 192)               150720

 dense_1 (Dense)             (None, 10)                1930

=================================================================
Total params: 152,650
Trainable params: 152,650
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
tuner.results_summary()
```

```
Results summary
Results in my_dir/helloworld
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x7f958d582a90>
Trial summary
Hyperparameters:
num_layers: 1
units_0: 192
activation: tanh
dropout: False
lr: 0.003449485159787074
units_1: 480
units_2: 64
Score: 0.9671000242233276
Trial summary
Hyperparameters:
num_layers: 2
units_0: 384
activation: tanh
dropout: False
lr: 0.0003271739205768162
units_1: 288
units_2: 160
Score: 0.9628500044345856
Trial summary
Hyperparameters:
num_layers: 3
units_0: 224
activation: tanh
dropout: False
lr: 0.005256055651516419
units_1: 32
units_2: 32
Score: 0.943450003862381
```

Choose best hyperparameters

In [22]:
```
best_hps = tuner.get_best_hyperparameters(5)
model = build_compile_model(best_hps[1])
```

```
number of layers
number of layers

Dropout is False
learning rate is 0.0003271739205768162
```

In [23]:
```
model.build(input_shape=(None, 28, 28))
```

In [24]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_3 (Dense)             (None, 384)               301440

 dense_4 (Dense)             (None, 288)               110880

 dense_5 (Dense)             (None, 10)                2890

=================================================================
Total params: 415,210
Trainable params: 415,210
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
best_hps[0].get('units_1')
```

Out[36]:
```
480
```

In [ ]: