mnist_optuna

March 8, 2021

[ ]:
```
!pip install optuna
```

[ ]:
```
import tensorflow as tf
import tensorflow.keras as keras
from keras.datasets import mnist
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,␣
↪BatchNormalization, Activation
import tensorflow.keras.backend as K
from tensorflow.keras import Sequential

import optuna
```

[ ]:
```
batch_size = 128
num_classes = 10
epochs = 12
```

[ ]:
```
img_rows, img_cols = 28, 28
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

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

[ ]:
```
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
```

[ ]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0
```

[ ]:
```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

[ ]:
```
params = {
    'conv_filters': [[16, 64], [16, 64], [16, 64]],
    'dense_filters': [[128, 256], [32, 128], [32, 128]]
```

```
}
```

[ ]:
```
def define_model(trial):

    n_layers_conv = trial.suggest_int('conv_layers', 0, 3)
    n_layers_dense = trial.suggest_int('dense_layers', 0, 3)
    model = Sequential()

    for i in range(n_layers_conv):
        filter_size = trial.suggest_int('conv'+str(i+1),␣
↪params['conv_filters'][i][0], params['conv_filters'][i][1])
        p = trial.suggest_float('conv_drop'+str(i+1), 0.0, 0.4)
        model.add(Conv2D(filter_size, [3, 3], padding='same', activation=None,␣
↪use_bias=False))
        model.add(BatchNormalization())
        model.add(Activation('relu'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(rate=p))

    model.add(Flatten())

    for i in range(n_layers_dense):
        neuron_size = trial.suggest_int('dense'+str(i+1),␣
↪params['dense_filters'][i][0], params['dense_filters'][i][1])
        p = trial.suggest_float('dense_drop'+str(i+1), 0.0, 0.4)
        model.add(Dense(neuron_size, use_bias=False))
        model.add(BatchNormalization())
        model.add(Activation('relu'))
        model.add(Dropout(rate=p))

    model.add(Dense(num_classes, activation='softmax'))

    return model
```

[ ]:
```
class OptunaReporter(keras.callbacks.Callback):

    def __init__(self, trial):
        self.trial = trial

    def on_epoch_end(self, epoch, logs=None):
        self.trial.report(logs['accuracy'], epoch)
```

[ ]:
```
def objective(trial):
    lr = trial.suggest_float('lr', 1e-5, 1e-1, log=True)
    optimizer = keras.optimizers.Adam(learning_rate=lr)
    model = define_model(trial)
```

```
    model.compile(loss=keras.losses.categorical_crossentropy,␣
↪optimizer=optimizer, metrics=['accuracy'])
    model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,␣
↪verbose=0, validation_data=(x_test, y_test),␣
↪callbacks=[OptunaReporter(trial)])
    score = model.evaluate(x_test, y_test, verbose=0)

    if trial.should_prune():
        raise optuna.exceptions.TrialPruned()

    return score[1]
```

[ ]:
```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

[ ]:
```
study.best_params, study.best_value
```

[ ]:
```
({'conv1': 44,
  'conv2': 56,
  'conv_drop1': 0.18567221804314119,
  'conv_drop2': 0.05041324298614322,
  'conv_layers': 2,
  'dense1': 212,
  'dense2': 78,
  'dense_drop1': 0.24997374242018003,
  'dense_drop2': 0.03918263915216241,
  'dense_layers': 2,
  'lr': 0.0018628251888028735},
 0.9941999912261963)
```

[ ]:
```
fig = optuna.visualization.plot_param_importances(study)
fig.show()
```

[ ]:
```

```