TL03 KerasTuner

Índice

1 Introducción

KerasTuner: librería para el ajuste de hiperparámetros

Documentación: https://keras.io/keras_tuner

Github: https://github.com/keras-team/keras-tuner

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

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

API: https://keras.io/api/keras_tuner

- **Hyperparameters:** especificación de hiperparámetros y valores
- **Tuners:** algoritmos de ajuste de hiperparámetros
- **Oracles**: algoritmos básicos de búsqueda de nuevos hiperparámetros
- **HyperModels:** espacios de búsqueda predefinidos para ciertas familias de modelos
- **Errors:** tipos de errores que pueden producirse durante el ajuste

2 API

In [ ]:
```
import keras_tuner
```

2.1 HyperParameters

Documentación: https://keras.io/api/keras_tuner/hyperparameters

Clase HyperParameters: contiene un espacio de hiperparámetros y sus valores actuales

Métodos:

- **Boolean:** escoge entre `True` y `False`
- **Choice:** escoge valor de un conjunto predefinido
- **Fixed:** valor fijo, no ajustable
- **Float:** devuelve número en coma flotante
- **Int:** devuelve número entero

In [ ]:
```
hp = keras_tuner.HyperParameters()
print(hp.Boolean("hpBool"))
print(hp.Choice("model_type", ["mlp", "cnn"]))
print(hp.Float("image_rotation_factor", min_value=0, max_value=1, step=0.2))
print(hp.Int("units", min_value=750, max_value=850, step=25))
```

```
False
mlp
0.0
750
```

2.2 Tuners

Documentación: https://keras.io/api/keras_tuner/tuners

Clase Tuner: clase base de búsqueda de hiperparámetros mediante creación, entrenamiento y evaluación de modelos

- Por cada experimento (trial), los nuevos valores de hiperparámetros a evaluar se obtienen de un objeto Oracle
- Tras ajustar el modelo con `model.fit(...)`, los resultados de la evaluación se devuelven al objeto Oracle

Métodos:

- **search_space_summary:** muestra un resumen del espacio de búsqueda
- **run_trial:** evalúa un conjunto de valores de hiperparámetros
- **search:** búsqueda de valores óptimos para los hiperparámetros
- **results_summary:** muestra un resumen de los resultados del ajuste
- **get_best_hyperparameters:** devuelve los mejores hiperparámetros
- **get_best_models:** devuelve los mejores modelos
- etc.

Subclases de Tuner:

- **Objective:** objetivo de la optimización durante el ajuste
- **RandomSearch:** tuner de búsqueda aleatoria
- **GridSearch:** tuner de búsqueda en rejilla
- **BayesianOptimization:** tuner de búsqueda con un proceso Gaussiano
- **Hyperband:** tuner basado en una variante del algoritmo HyperBand
- **SklearnTuner:** tuner para modelos scikit-learn

2.3 Oracles

Documentación: https://keras.io/api/keras_tuner/oracles

Clase Oracle: clase base de algoritmos de búsqueda de nuevos valores de hiperparámetros

- Los nuevos valores se generan a partir de resultados de evaluación de valores previos (obtenidos por un objeto Tuner)

Subclases de Oracle:

- **RandomSearch:** oráculo de búsqueda aleatoria
- **GridSearch:** oráculo de búsqueda en rejilla
- **BayesianOptimization:** oráculo de búsqueda con un proceso Gaussiano
- **Hyperband:** oráculo basado en una variante del algoritmo HyperBand

2.4 HyperModels

Documentación: https://keras.io/api/keras_tuner/hypermodels

Clase HyperModel: define un espacio específico de búsqueda de modelos

Subclases de HyperModel: HyperEfficientNet, HyperImageAugment, HyperResNet, HyperXception

2.5 Errors

Documentación: https://keras.io/api/keras_tuner/errors

Clases: FailedTrialError, FatalError, FatalValueError, FatalTypeError, FatalRuntimeError

3 Ajuste de la arquitectura

Ejemplo ilustrativo: ajuste de la arquitectura del MLP para MNIST visto en la sesión anterior

Inicialización: librerías y semilla para la generación de números aleatorios

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
import keras_tuner
keras.utils.set_random_seed(23)
```

Lectura y partición train-val-test de MNIST: reservamos las últimas 10000 muestras del training para validación

In [ ]:
```
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.mnist.load_data()

input_dim = 784
x_train_val = x_train_val.reshape(-1, input_dim).astype("float32") / 255.0
x_test = x_test.reshape(-1, input_dim).astype("float32") / 255.0

num_classes = 10
y_train_val = keras.utils.to_categorical(y_train_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
```

Definición del espacio de búsqueda: para ajustar el número de capas ocultas y unidades de cada una

In [ ]:
```
def build_model(hp):
    M = keras.Sequential()
    M.add(keras.Input(shape=(784,)))
    for L in range(hp.Int("num_layers", 1, 3)):
        M.add(keras.layers.Dense(
            units=hp.Int(f"units_{L}", min_value=700, max_value=900, step=100),
            activation='relu'))
    M.add(keras.layers.Dense(10, activation='softmax'))
    M.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return M
```

Definición del tuner: 10 experimentos con BayesianOptimization para optimizar la precisión en validación

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    hypermodel=build_model, # función para construir un modelo
    objective="val_accuracy", # objetivo a optimizar
    max_trials=10, # máximo número de experimentos (trials) a realizar
    executions_per_trial=1, # modelos a crear, entrenar y evaluar por cada trial
    overwrite=True, # para reinicializar o continuar una búsqueda previa
    directory="/tmp", # directorio para guardar los resultados del ajuste
    project_name="MNIST", # subdirectorio dentro de directory
)
```

Resumen del espacio de búsqueda:

In [ ]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 2
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': 'linear'}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 700, 'max_value': 900, 'step': 100, 'sampling': 'linear'}
```

Búsqueda de valores óptimos de hiperparámetros:

In [ ]:
```
tuner.search(x_train, y_train, batch_size=16, epochs=10, validation_data=(x_val, y_val))
```

```
Trial 10 Complete [00h 00m 23s]
val_accuracy: 0.978600025177002

Best val_accuracy So Far: 0.9817000031471252
Total elapsed time: 00h 04m 22s
```

Resumen de los resultados del ajuste: limitado al mejor modelo en validación

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 06 summary
Hyperparameters:
num_layers: 1
units_0: 800
units_1: 800
units_2: 700
Score: 0.9817000031471252
```

Evaluación: del mejor modelo en validación

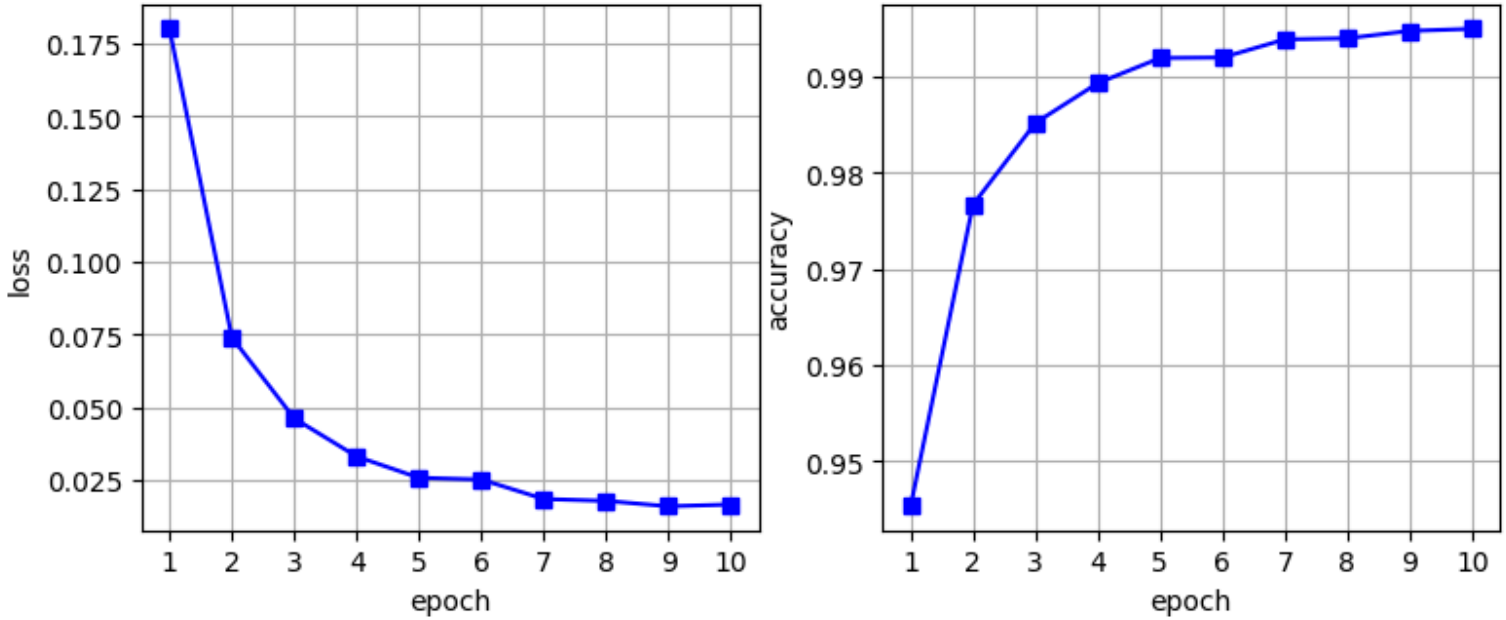
In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.109
Precisión: 98.2%
```

Entrenamiento con train-val: usando valores óptimos de hiperparámetros

In [ ]:
```
M = build_model(tuner.get_best_hyperparameters(1)[0])
H = M.fit(x_train_val, y_train_val, batch_size=16, epochs=10, verbose=0)
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss')
ax.set_xticks(xx); ax.plot(xx, H.history['loss'], color='b', marker='s')
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.set_xticks(xx); ax.plot(xx, H.history['accuracy'], color='b', marker='s');
```



Evaluación: del modelo entrenado con train-val y valores óptimos de hiperparámetros

In [ ]:
```
score = M.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.1062
Precisión: 98.2%
```

4 Ejercicio: Fashion-MNIST

Ejercicio: realiza un experimento similar al de MNIST con Fashion-MNIST

Inicialización: librerías y semilla para la generación de números aleatorios

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
import keras_tuner
keras.utils.set_random_seed(23)
```

Lectura y partición train-val-test de Fashion-MNIST: reservamos las últimas 10000 muestras del training para validación

In [ ]:
```
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

input_dim = 784
x_train_val = x_train_val.reshape(-1, input_dim).astype("float32") / 255.0
x_test = x_test.reshape(-1, input_dim).astype("float32") / 255.0

num_classes = 10
y_train_val = keras.utils.to_categorical(y_train_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
```

Definición del espacio de búsqueda: para ajustar el número de capas ocultas y unidades de cada una

In [ ]:
```
def build_model(hp):
    M = keras.Sequential()
    M.add(keras.Input(shape=(784,)))
    for L in range(hp.Int("num_layers", 1, 3)):
        M.add(keras.layers.Dense(
            units=hp.Int(f"units_{L}", min_value=700, max_value=900, step=100),
            activation='relu'))
    M.add(keras.layers.Dense(10, activation='softmax'))
    M.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return M
```

Definición del tuner: 10 experimentos con BayesianOptimization para optimizar la precisión en validación

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    hypermodel=build_model, # función para construir un modelo
    objective="val_accuracy", # objetivo a optimizar
    max_trials=10, # máximo número de experimentos (trials) a realizar
    executions_per_trial=1, # modelos a crear, entrenar y evaluar por cada trial
    overwrite=True, # para reinicializar o continuar una búsqueda previa
    directory="/tmp", # directorio para guardar los resultados del ajuste
    project_name="Fashion-MNIST", # subdirectorio dentro de directory
)
```

Resumen del espacio de búsqueda:

In [ ]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 2
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': 'linear'}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 700, 'max_value': 900, 'step': 100, 'sampling': 'linear'}
```

Búsqueda de valores óptimos de hiperparámetros:

In [ ]:
```
tuner.search(x_train, y_train, batch_size=16, epochs=20, validation_data=(x_val, y_val))
```

```
Trial 10 Complete [00h 00m 45s]
val_accuracy: 0.8824999928474426

Best val_accuracy So Far: 0.8901000022888184
Total elapsed time: 00h 08m 39s
```

Resumen de los resultados del ajuste: limitado al mejor modelo en validación

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/Fashion-MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 06 summary
Hyperparameters:
num_layers: 1
units_0: 800
units_1: 800
units_2: 700
Score: 0.8901000022888184
```

Evaluación: del mejor modelo en validación

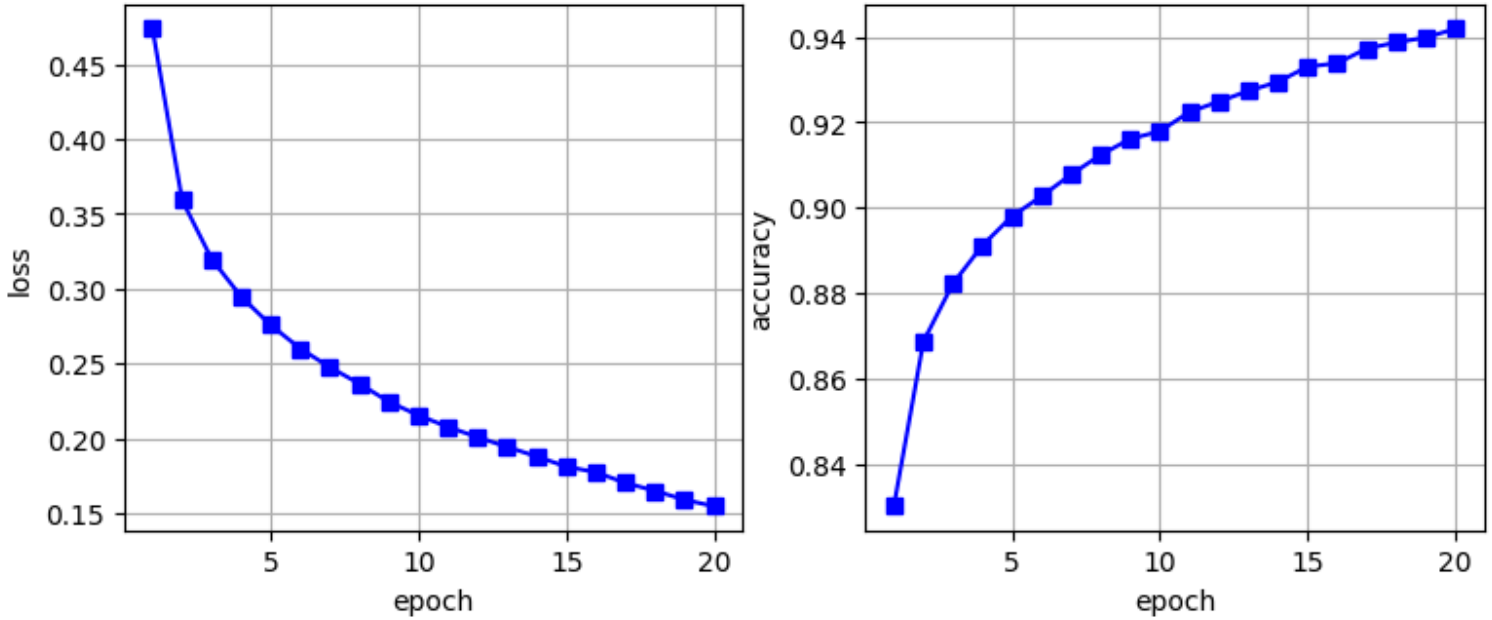
In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.4696
Precisión: 88.0%
```

Entrenamiento con train-val: usando valores óptimos de hiperparámetros

In [ ]:
```
M = build_model(tuner.get_best_hyperparameters(1)[0])
H = M.fit(x_train_val, y_train_val, batch_size=16, epochs=20, verbose=0)
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss')
ax.plot(xx, H.history['loss'], color='b', marker='s')
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', marker='s');
```



Evaluación: del modelo entrenado con train-val y valores óptimos de hiperparámetros

In [ ]:
```
score = M.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.4642
Precisión: 88.3%
```