Introduction to the Keras Tuner

Overview

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called *hyperparameter tuning* or *hypertuning*.

Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

1. **Model hyperparameters** which influence model selection such as the number and width of hidden layers
2. **Algorithm hyperparameters** which influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent (SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier

In this tutorial, you will use the Keras Tuner to perform hypertuning for an image classification application.

Setup

```
import tensorflow as tf
from tensorflow import keras
```

```
2024-08-16 01:25:04.811063: E external/local_xla/xla/stream_executor/cuda/cuda
2024-08-16 01:25:04.832191: E external/local_xla/xla/stream_executor/cuda/cuda
2024-08-16 01:25:04.838460: E external/local_xla/xla/stream_executor/cuda/cuda
```

Install and import the Keras Tuner.

```
$ pip install -q -U keras-tuner
```

```
import keras_tuner as kt
```

Download and prepare the dataset

In this tutorial, you will use the Keras Tuner to find the best hyperparameters for a machine learning model that classifies images of clothing from the Fashion MNIST dataset (https://github.com/zalandoresearch/fashion-mnist).

Load the data.

```
(img_train, label_train), (img_test, label_test) = keras.datasets.fashion_mnis
```

```
# Normalize pixel values between 0 and 1
img_train = img_train.astype('float32') / 255.0
img_test = img_test.astype('float32') / 255.0
```

Define the model

When you build a model for hypertuning, you also define the hyperparameter search space in addition to the model architecture. The model you set up for hypertuning is called a *hypermodel*.

You can define a hypermodel through two approaches:

- By using a model builder function
- By subclassing the `HyperModel` class of the Keras Tuner API

You can also use two pre-defined HyperModel (https://keras.io/api/keras_tuner/hypermodels/) classes - HyperXception (https://keras.io/api/keras_tuner/hypermodels/hyper_xception/) and HyperResNet (https://keras.io/api/keras_tuner/hypermodels/hyper_resnet/) for computer vision applications.

In this tutorial, you use a model builder function to define the image classification model. The model builder function returns a compiled model and uses hyperparameters you define inline to hypertune the model.

```
def model_builder(hp):
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_shape=(28, 28)))

  # Tune the number of units in the first Dense layer
  # Choose an optimal value between 32-512
  hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
  model.add(keras.layers.Dense(units=hp_units, activation='relu'))
  model.add(keras.layers.Dense(10))

  # Tune the learning rate for the optimizer
  # Choose an optimal value from 0.01, 0.001, or 0.0001
  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
                metrics=['accuracy'])

  return model
```

Instantiate the tuner and perform hypertuning

Instantiate the tuner to perform the hypertuning. The Keras Tuner has four tuners available - `RandomSearch`, `Hyperband`, `BayesianOptimization`, and `Sklearn`. In this tutorial, you use the Hyperband (https://arxiv.org/pdf/1603.06560.pdf) tuner.

To instantiate the Hyperband tuner, you must specify the hypermodel, the **`objective`** to optimize and the maximum number of epochs to train (**`max_epochs`**).

```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='intro_to_kt')
```

```
WARNING: All log messages before absl::InitializeLog() is called are written
I0000 00:00:1723771509.637777   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.641612   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.644868   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.648549   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.660168   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.663655   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.666633   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.670142   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.673591   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.677149   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.680118   14090 cuda_executor.cc:1015] successful NUMA
I0000 00:00:1723771509.683613   14090 cuda_executor.cc:1015] successful NUMA
```

The Hyperband tuning algorithm uses adaptive resource allocation and early-stopping to quickly converge on a high-performing model. This is done using a sports championship style bracket. The algorithm trains a large number of models for a few epochs and carries forward only the top-performing half of models to the next round. Hyperband determines the number of models to train in a bracket by computing $1 + \log_{\texttt{factor}}(\texttt{max\_epochs})$ and rounding it up to the nearest integer.

Create a callback to stop training early after reaching a certain value for the validation loss.

```
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

Run the hyperparameter search. The arguments for the search method are the same as those used for **`tf.keras.model.fit`** in addition to the callback above.

```
tuner.search(img_train, label_train, epochs=50, validation_split=0.2, callbac

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal number of units in the firs
layer is {best_hps.get('units')} and the optimal learning rate for the optimiz
is {best_hps.get('learning_rate')}.
""")
```

```
Trial 30 Complete [00h 00m 25s]
val_accuracy: 0.8913333415985107

Best val_accuracy So Far: 0.8913333415985107
Total elapsed time: 00h 05m 37s

The hyperparameter search is complete. The optimal number of units in the firs
layer is 416 and the optimal learning rate for the optimizer
is 0.001.
```

Train the model

Find the optimal number of epochs to train the model with the hyperparameters obtained from the search.

```
# Build the model with the optimal hyperparameters and train it on the data fo
model = tuner.hypermodel.build(best_hps)
history = model.fit(img_train, label_train, epochs=50, validation_split=0.2)

val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))
```

```
Epoch 1/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7774 - loss: 0.6344 -
Epoch 2/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8643 - loss: 0.3766 -
Epoch 3/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8790 - loss: 0.3291 -
Epoch 4/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8884 - loss: 0.3036 -
Epoch 5/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8955 - loss: 0.2840 -
Epoch 6/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8998 - loss: 0.2655 -
Epoch 7/50
```

Re-instantiate the hypermodel and train it with the optimal number of epochs from above.

```
hypermodel = tuner.hypermodel.build(best_hps)

# Retrain the model
hypermodel.fit(img_train, label_train, epochs=best_epoch, validation_split=0.2
```

```
Epoch 1/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7789 - loss: 0.6225 -
Epoch 2/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8678 - loss: 0.3664 -
Epoch 3/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8781 - loss: 0.3343 -
Epoch 4/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8879 - loss: 0.3070 -
Epoch 5/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8908 - loss: 0.2902 -
Epoch 6/32
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9002 - loss: 0.2703 -
Epoch 7/32
```

To finish this tutorial, evaluate the hypermodel on the test data.

```
eval_result = hypermodel.evaluate(img_test, label_test)
print("[test loss, test accuracy]:", eval_result)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8873 - loss: 0.4595
[test loss, test accuracy]: [0.4649185538291931, 0.8881000280380249]
```

The `my_dir/intro_to_kt` directory contains detailed logs and checkpoints for every trial (model configuration) run during the hyperparameter search. If you re-run the hyperparameter search, the Keras Tuner uses the existing state from these logs to resume the search. To disable this behavior, pass an additional `overwrite=True` argument while instantiating the tuner.

Summary

In this tutorial, you learned how to use the Keras Tuner to tune hyperparameters for a model. To learn more about the Keras Tuner, check out these additional resources:

- Keras Tuner on the TensorFlow blog (https://blog.tensorflow.org/2020/01/hyperparameter-tuning-with-keras-tuner.html)
- Keras Tuner website (https://keras-team.github.io/keras-tuner/)

Also check out the HParams Dashboard (https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams) in TensorBoard to interactively tune your model hyperparameters.