Hyperparameters tuning
Keras tuner implementation
Lecture 9

Course of:

Signal and imaging acquisition and modelling in environment

05/04/2024

Federico De Guio - Matteo Fossati

Keras Sequential model with Hyperparameter tuning

```
!pip install keras-tuner
import keras_tuner as kt
```

Hyperparameter tuning looks for the best parameters in your CNN implementation. This is done by optimizing a metric based on the **validation sample.**

We can search for the best values in a dynamical model for the following parameters:

- Integer hyperparameter with hp.Int()
- Which activation function to use with hp.Choice()
- Float hyperparameters (e.g. the learning rate) with hp.Float()
- Add or remove layers with a boolean choice function with hp.Boolean()

Keras Sequential model with Hyperparameter tuning

First we define a dynamic model

```
[ ] # Define architecture for model
    def build_model(hp):

      model = Sequential()

      hp_kernel_1 = hp.Int('kernel1', min_value=4, max_value=10, step=2)
      hp_kernel_size_1 = hp.Int('kernel_size1', min_value=3, max_value=11, step=2)
      model.add(Conv2D(hp_kernel_1, (hp_kernel_size_1, hp_kernel_size_1), activation='relu', strides=(1, 1), padding='same'))
      model.add(BatchNormalization())
      model.add(MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid'))
      model.add(Dropout(0.5))

      hp_kernel_2 = hp.Int('kernel2', min_value=8, max_value=20, step=2)
      hp_kernel_size_2 = hp.Int('kernel_size2', min_value=3, max_value=11, step=2)
      model.add(Conv2D(hp_kernel_2, (hp_kernel_size_2, hp_kernel_size_2), activation='relu', strides=(1, 1), padding='same'))
      model.add(BatchNormalization())
      model.add(MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid'))
      model.add(Dropout(0.5))

      hp_kernel_3 = hp.Int('kernel3', min_value=16, max_value=40, step=2)
      hp_kernel_size_3 = hp.Int('kernel_size3', min_value=3, max_value=11, step=2)
      model.add(Conv2D(hp_kernel_3, (hp_kernel_size_3, hp_kernel_size_3), activation='relu', strides=(1, 1), padding='same'))
      model.add(BatchNormalization())
      model.add(MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid'))
      model.add(Dropout(0.5))

      model.add(Flatten())
      model.add(Dense(64, activation='softmax', kernel_regularizer=l2(0.0001)))
      model.add(Dense(32, activation='softmax', kernel_regularizer=l2(0.0001)))
      model.add(Dense(1, activation='sigmoid'))

      lr = hp.Choice("learning_rate",values=[1e-1, 1e-2, 1e-3])
      model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics='accuracy')

      return model
```

Keras Sequential model with Hyperparameter tuning

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

```
nb_epoch = 100
batch_size = 128
shuffle = True

#Define an early stopping condition
stop_early = EarlyStopping(monitor='val_loss',patience=10)

hyperpar_names = ['kernel1', 'kernel_size1','kernel2', 'kernel_size2','kernel3', 'kernel_size3', 'learning_rate']
```

```
tuner = kt.RandomSearch(build_model, objective='val_loss', max_trials=25,  project_name='GALCNN_RandomSrc')
tuner.search_space_summary()
```

```
Reloading Tuner from ./GALCNN_RandomSrc/tuner0.json
Search space summary
Default search space size: 7
kernel1 (Int)
{'default': None, 'conditions': [], 'min_value': 4, 'max_value': 10, 'step': 2, 'sampling': 'linear'}
kernel_size1 (Int)
{'default': None, 'conditions': [], 'min_value': 3, 'max_value': 11, 'step': 2, 'sampling': 'linear'}
kernel2 (Int)
{'default': None, 'conditions': [], 'min_value': 8, 'max_value': 20, 'step': 2, 'sampling': 'linear'}
kernel_size2 (Int)
{'default': None, 'conditions': [], 'min_value': 3, 'max_value': 11, 'step': 2, 'sampling': 'linear'}
kernel3 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 40, 'step': 2, 'sampling': 'linear'}
kernel_size3 (Int)
{'default': None, 'conditions': [], 'min_value': 3, 'max_value': 11, 'step': 2, 'sampling': 'linear'}
learning_rate (Choice)
{'default': 0.1, 'conditions': [], 'values': [0.1, 0.01, 0.001], 'ordered': True}
```

```
#Tuner settings
```

```
#Run the tuner
tuner.search(X_train, y_train, epochs=nb_epoch, batch_size=batch_size,
             shuffle=shuffle, validation_data=(X_valid, y_valid), callbacks=[stop_early])
```

Keras Sequential model with Hyperparameter tuning

```
best_hyband = hptuner.get_best_hyperparameters()[0]

for pp in hyperpar_names:
    print('Best Value for parameter {} : {}'.format(pp,best_hyband.get(pp)))
```

```
Best Value for parameter kernel1 : 10
Best Value for parameter kernel_size1 : 11
Best Value for parameter kernel2 : 18
Best Value for parameter kernel_size2 : 5
Best Value for parameter kernel3 : 22
Best Value for parameter kernel_size3 : 9
Best Value for parameter learning_rate : 0.001
```

```
#Build the best model
HYbest = hptuner.hypermodel.build(best_hyband)
```

Other Tuners

- **RandomSearch**
  - It doesn't learn from previously tested parameter combinations, and samples parameter combinations from a search space randomly
- **BayesianOptimization**
  - Doesn't sample hyperparameter combinations randomly, it follows a probabilistic approach under the hood. This approach takes into account already tested combinations and uses this information to sample the next combination for a test
- **Hyperband**
  - Optimized version of RandomSearch. 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 $1 + \log_{factor}(\text{max\_epochs})$ and rounding it up to the nearest integer.

Implement Hyperparameter tuning in your CNN exercise from Lecture 8.

Use the tuner you prefer but be aware Hyperband is computationally expensive!