Deep Learning

Optimizing Hyperparameters

POLITÉCNICO DE LEIRIA

Slides by Carlos Grilo & Rolando Miragaia

Optimizing hyperparameters

- Until now, for each model that we have developed, we have defined just one model architecture and training hyperparameters combination
- However, we usually don't know what the best combination is
- So, we need to engage in a search process trying to find the best combination
- Done "by hand", this is basically a trial-and-error process that can be very boring and time consuming

Hyperparameter optimization tools

- Happily, there are tools that we can use to automate the search process
- These tools allow us to define a range of values for each hyperparameter that we want to include as part of the search process
- Example of hyperparameter optimization tools (there are others):
  - Optuna, https://optuna.org/; https://optuna.readthedocs.io/en/stable/index.html
  - KerasTuner, https://keras.io/guides/keras_tuner/

Using Optuna

Optuna

- Optuna is an open source hyperparameter optimization framework
- It is framework agnostic: we can use it with any machine learning or deep learning framework
- Installation (command line):

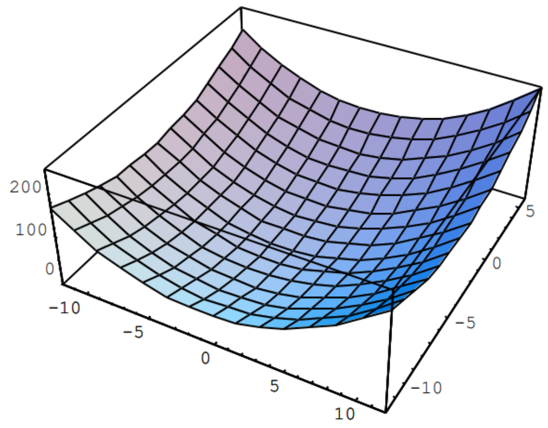
```
$ pip install optuna
```

Objective function

- In order to use Optuna, we need to define a function to be optimized

- Here is an example:

```
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    y = trial.suggest_float("y", -10, 10)
  return x ** 2 + y ** 2 + 2 * x + 8 * y
```



- This function returns the value of $x^2 + y^2 + 2x + 8y$ for some combination of x and y

- Our goal is to find the combination (x, y) that minimizes the output of the objective function

- During the optimization process, Optuna repeatedly calls and evaluates the objective function with different combinations of x and y

The trial object

```
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    y = trial.suggest_float("y", -10, 10)
    return x ** 2 + y ** 2 + 2 * x + 8 * y
```

- A trial object corresponds to a single execution of the objective function
- It is internally instantiated upon each invocation of the function

The `suggest` API

```
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    y = trial.suggest_float("y", -10, 10)
    return x ** 2 + y ** 2 + 2 * x + 8 * y
```

- The `suggest` API functions (for example, `suggest_float()`) are called inside the objective function to obtain parameters for a trial
- `suggest_float()` selects parameters uniformly within the range provided. In our example, from to -10 to 10, for both x and y

The `suggest` API

- Suggesting a value for a categorical parameter:

```
kernel = trial.suggest_categorical("kernel", ["linear", "poly", "rbf"])
```

- Suggesting a value for a floating point parameter

```
momentum = trial.suggest_float("momentum", 0.0, 1.0)

lr = trial.suggest_float(“lr", 1e-5, 1e-3, log=True)
```

- Suggesting a value for an integer parameter:

```
n_estimators = trial.suggest_int("n_estimators", 50, 400)
```

Starting the optimization process

- To start the optimization process, we create a study object and pass the objective function to method `optimize()` as follows:

```
study = optuna.create_study()
study.optimize(objective, n_trials=100)
```

Objective function

Number of trials

- Method `optimize()` calls the objective function as many times as the defined number of trials

- Each call, a different combination of the parameters to be optimized is generated, according to the optimization algorithm that is used

Getting the best parameters

- We can get the best parameters combination as follows:

```
best_params = study.best_params
found_x = best_params["x"]
found_y = best_params["y"]
print("Found x: {}".format(found_x))
print("Found y: {}".format(found_y))
print("Found f(x, y): {}".format(found_x ** 2 + found_y ** 2 + 2 * found_x + 8 * found_y))
```

Example with the MNIST dataset

Creating the notebook

- Create a new notebook named `05_Optuna_01_MNIST.ipynb`

- Add the following cell code to mount your Google Drive on Colab:

```
from google.colab import drive
drive.mount('/content/drive')
```

- Install Optuna

```
!pip install optuna
```

Loading and preparing the data

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

val_images = train_images[50000:]
val_labels = train_labels[50000:]

train_images = train_images[:50000]
train_labels = train_labels[:50000]

print(train_images.shape)
print(val_images.shape)
print(test_images.shape)
```

- In the hyperparameters optimization process we will use a validation set to assess the performance of the model
- We will use the loss value computed for the validation set to guide the optimization process
- In this example, we will use 50000 samples for training, 10000 samples for validation and 10000 samples for testing

The objective function

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import numy as np

def objective(trial):

  opt_num_hidden_dense_units = trial.suggest_int("opt_num_hidden_dense_units", 10, 100)
  opt_lr = trial.suggest_float("opt_lr", 1e-6, 1e-2, log=True)
  opt_bs = trial.suggest_int("opt_bs", 16, 128)
  #…
```

- In this example, we try to find the best combination of the following parameters:
  - Nº of units of the hidden dense layer of the CNN
  - Learning rate
  - Batch size

Continues next slide…

The objective function - continued

Model definition…

```
#…
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(opt_num_hidden_dense_units, activation="relu")(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
#…
```

Continues next slide…

The objective function - continued

```
#...
model.compile(
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=opt_lr),
    loss='categorical_crossentropy',
    metrics=['accuracy'])

history = model.fit(
    train_images,
    train_labels,
    epochs=5,
    validation_data=(val_images, val_labels),
    batch_size=opt_bs)

min_val_loss = np.amin(history.history["val_loss"])

return min_val_loss
```

The validation loss is used to guide the optimization process

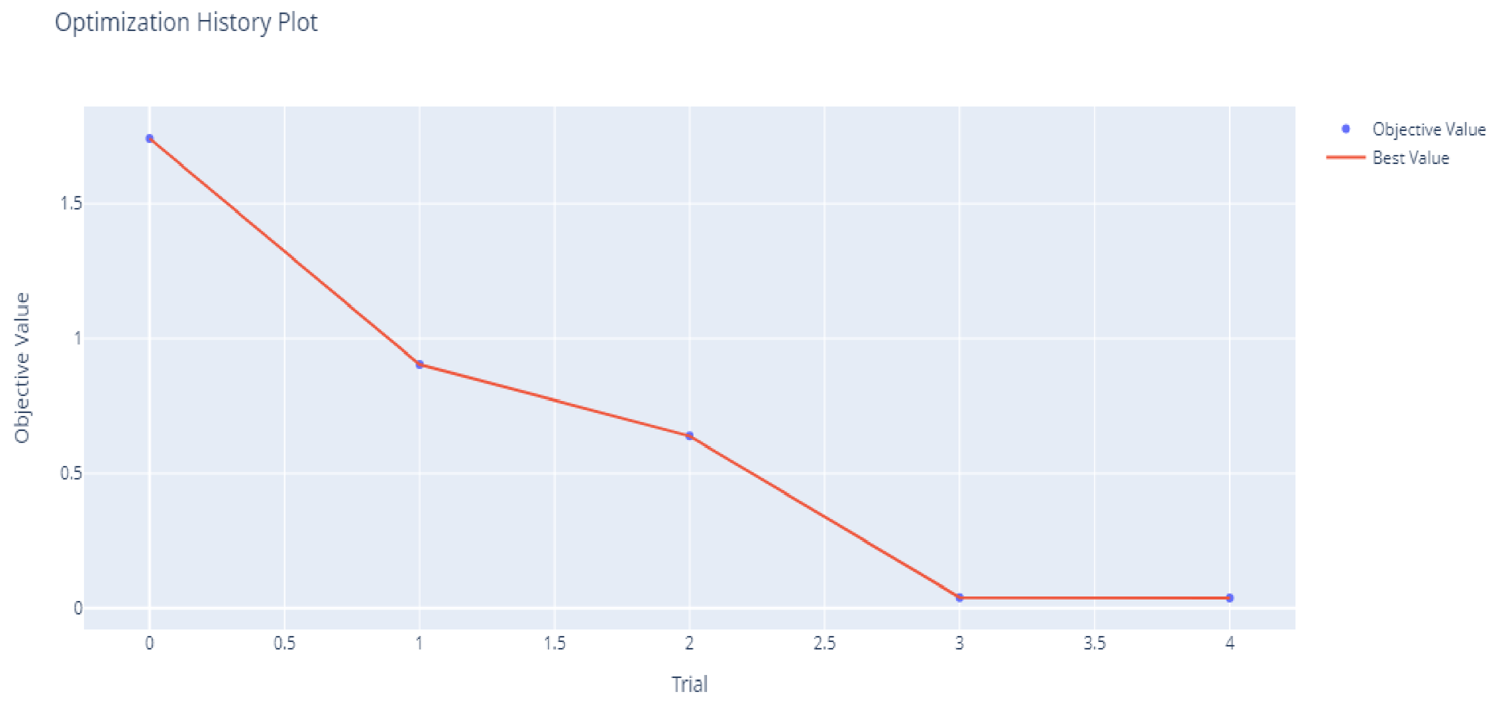
Starting the optimization process

```
import optuna as opt

study = opt.create_study()
study.optimize(objective, n_trials=5)
```

Usually, we use much more than 5 trials...

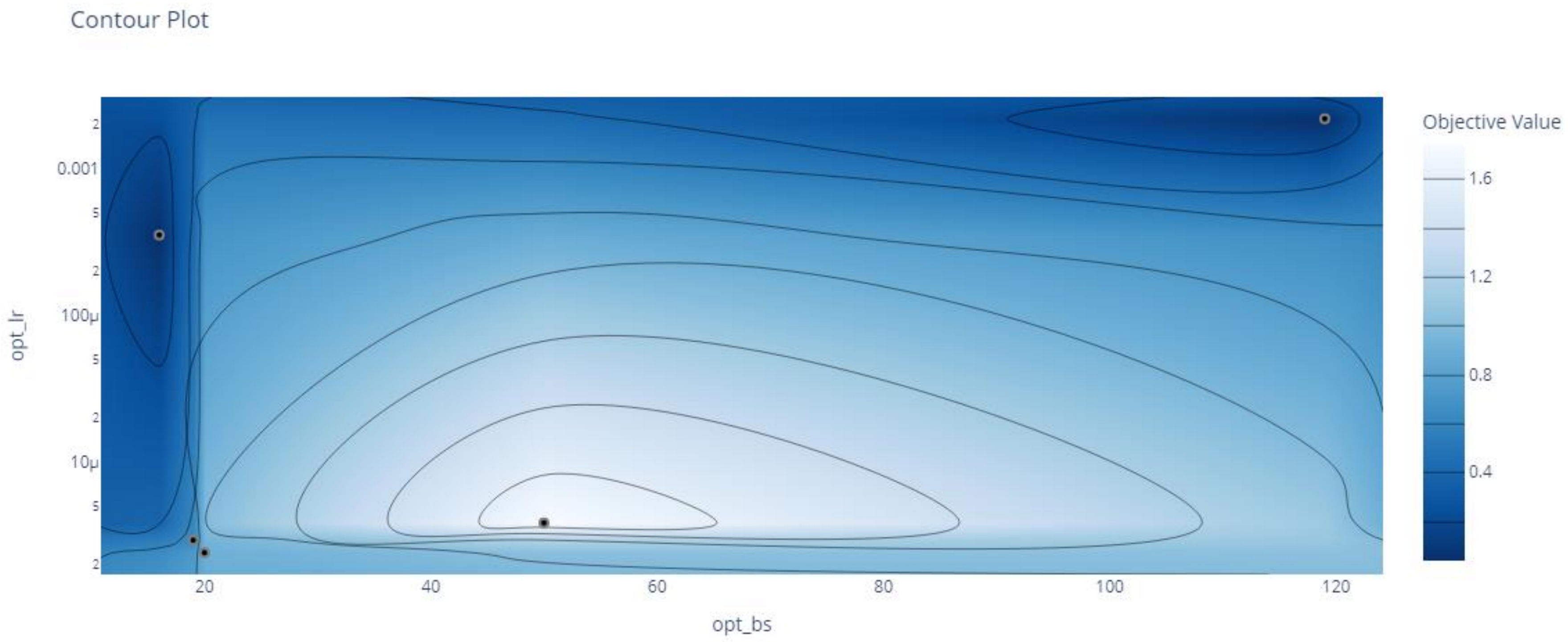
Visualizing the optimization history

```python
from optuna.visualization import plot_optimization_history

plot_optimization_history(study)
```



Contour plot

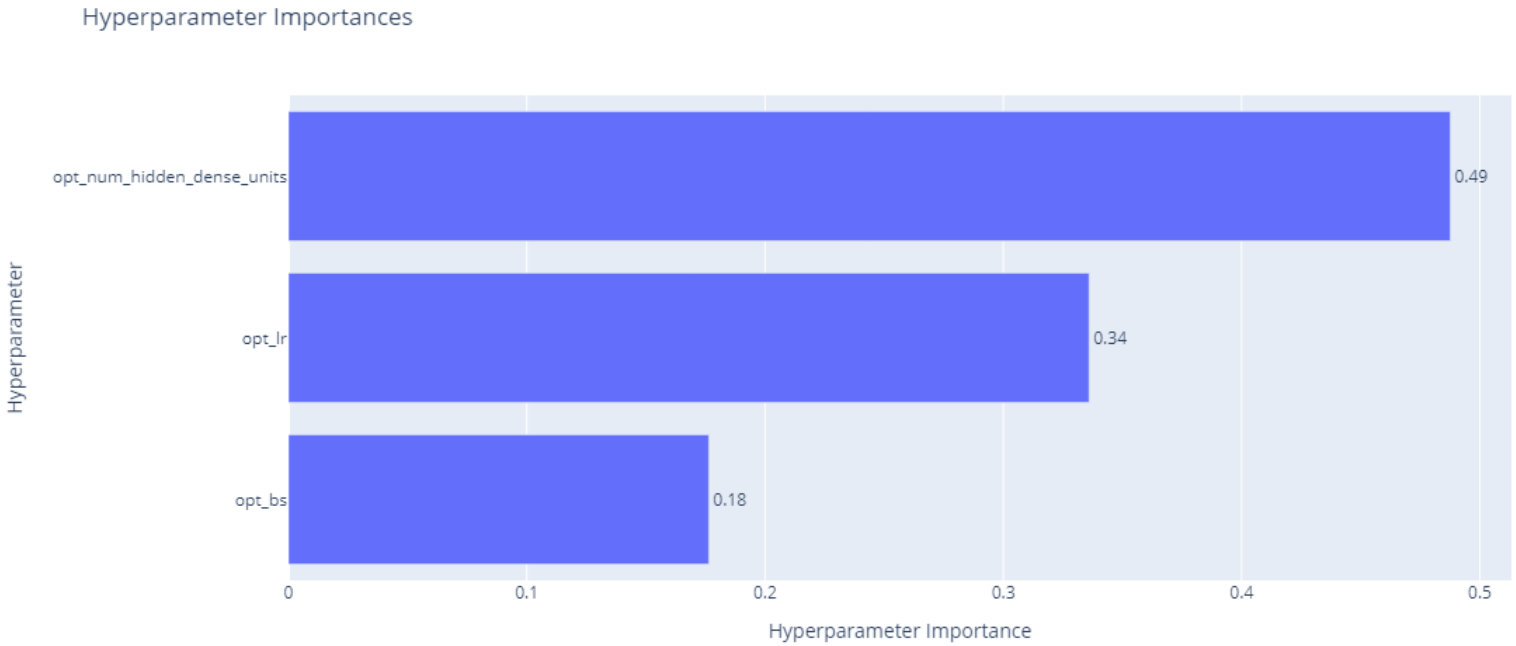
```
from optuna.visualization import plot_contour

plot_contour(study, params=["opt_lr", "opt_bs"])
```



- The `plot_contour()` function allow us to analyse what are the most promising regions
- We can combine different pair of parameters (try it)
- We can also include more than two parameters (try it)

Hyperparameters importances

```
from optuna.visualization import plot_param_importances

plot_param_importances(study)
```



Getting the best parameters

```
best_params = study.best_params

found_opt_num_hidden_dense_units = best_params["opt_num_hidden_dense_units"]
found_opt_lr = best_params["opt_lr"]
found_opt_bs = best_params["opt_bs"]

print("Found num hidden dense units: {}".format(found_opt_num_hidden_dense_units))
print("Found learning rate: {}".format(found_opt_lr))
print("Found batch size: {}".format(found_opt_bs))
```

Training the final model

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(found_opt_num_hidden_dense_units, activation="relu")(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=found_opt_lr),
    loss='categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5, batch_size=found_opt_bs)
```

- We now train the final model using the combination of parameters values return by Optuna

Testing and saving the final model

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc


-------


model.save('/content/drive/MyDrive/models/05_Optuna_01_MNIST.h5')
```

Example with the MNIST dataset

The end

Branches

- We can use conditionals
- In the code below, we can suggest values for the `svc_c` parameter or for the `rf_max_depth` parameter depending on the value suggested for parameter `classifier_name`

```
import sklearn.ensemble
import sklearn.svm

def objective(trial):
    classifier_name = trial.suggest_categorical("classifier_name", ["SVC", "RandomForest"])
    if classifier_name == "SVC":
        svc_c = trial.suggest_float("svc_c", 1e-10, 1e10, log=True)
        classifier_obj = sklearn.svm.SVC(C=svc_c)
    else:
        rf_max_depth = trial.suggest_int("rf_max_depth", 2, 32, log=True)
        classifier_obj = sklearn.ensemble.RandomForestClassifier(max_depth=rf_max_depth)
```

Loops

- We can use loops
- In the code below, the `n_layers` parameter defines the number of layers. Then, some number of units is suggested for each layer (`n_units`)

```
import torch
import torch.nn as nn


def create_model(trial, in_size):
    n_layers = trial.suggest_int("n_layers", 1, 3)

    layers = []
    for i in range(n_layers):
        n_units = trial.suggest_int("n_units_l{}".format(i), 4, 128, log=True)
        layers.append(nn.Linear(in_size, n_units))
        layers.append(nn.ReLU())
        in_size = n_units
    layers.append(nn.Linear(in_size, 10))

    return nn.Sequential(*layers)
```