Introduction

- The following notebook focuses on applying Optuna in Reinforcement Learning.
- Source / Reference of this notebook: https://colab.research.google.com/github/araffin/tools-for-robotic-rl-icra2022/blob/main/notebooks/optuna_lab.ipynb#scrollTo=4UU17YpjymPr
- To study the application of Optuna in Deep Learning, refer to: https://www.geeksforgeeks.org/hyperparameter-tuning-with-optuna-in-pytorch/

Quick guides on the step needed.

1. Define a config.
2. Define a search space or a function that returns the parameters that define the models.
3. Define an objective score function that will return the objective score function for the sampled set of hyperparameter.
4. Create a optimization loop
    - For each trial:
        - a) sample a hyperparameter
        - b) Use the hyperparameter to train. At intervals, evaluate the performance of the models and decided if to prune this trial.

Step 0: Library Installation and Import

```
# Install optuna library
!pip install optuna
```

Show hidden output

```
# Install Stable Baseline 3
!pip install stable-baselines3==2.0.0a5
```

Show hidden output

```
# Install gymnasium

!pip install swig
!pip install gymnasium[box2d]
```

Show hidden output

```
import optuna
from optuna.pruners import MedianPruner
from optuna.samplers import TPESampler
from optuna.visualization import plot_optimization_history, plot_param_importances


from stable_baselines3.common.callbacks import EvalCallback
from stable_baselines3.common.env_util import make_vec_env
from stable_baselines3.a2c import A2C
from stable_baselines3 import PPO

import gymnasium as gym

import torch as th
import torch.nn as nn
```

DEMO: Tune a A2C agent that plays CartPole-v1

Step 1: Create config / key parameters

- Terminology:

1. `TRIALS` - Each `TRIAL` is initiated with different sampled set of hyperparameter. If needed, multiple `JOBS` can be initiated in parallel for each trial. Each `TRIAL` will involve training the agent / model for `N_TIMESTEPS`.
2. `EVAL_EPISODES` - During the training that last for `N_TIMESTEPS`, at an interval of `EVAL_FREQ`, evaluation will be performed. For each evaluation, `N_EVAL_EPISODES` of evaluation episodes will be sampled and reviewed. This may help the scheduler to decide whether to

prune early.

```
# Config
# Hyperparameter Optimization Loop
N_TRIALS = 100 # Maximum number of trials during Hyperparameter Optimization Loop
N_JOBS = 1 # Number of parallel jobs to run during each trials in Hyperparameter Optimization Loop
N_STARTUP_TRIALS = 5 # Number of trials to perform random sampling (without relying on sampler) during the Hypeparameter Optimizatic
TIMEOUT = int(60*15) # Maximum number of times (in seconds) the entire loop is allowed up to.

# Evaluation Parameter for each set of hyperparameter
N_TIMESTEPS = int(2e4) # Training budget - Number of time steps in one FULL TRIAL for each set of hyperparameter.
N_EVALUATIONS = 2 # Number of intermediate evaluations performed in one FULL TRIAL for each set of hyperparameter.
EVAL_FREQ = int(N_TIMESTEPS / N_EVALUATIONS) # Step interval for each intermediate evaluations during one FULL TRIAL.
N_EVAL_EPISODES = 10 # Number of episodes to be sampled for each evaluation.


# Environment Parameter
N_EVAL_ENVS = 5 # Number of environments used in parallel during evaluation.

ENV_ID = "CartPole-v1" # ID of the environments, to be initiated with gym.make()

# Algorithm Parameter
ALGO_NAME = "A2C"
```

Step 2: Define the search space

```
def sample_a2c_params(trial):

  """
  Sample a set of hyperparameter to be trial'd.

  Args:
    trial (optuna.Trial) : An Optuna trial object.

  Returns:
    params (dict): The hyperparameters to be trial'd. Its key matches the keywords used in defining models.
  """

  # Refer to this link: https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html#example
  # To study the hyperparameter to be updated.

  ###################
  # Discount factor #
  ###################
  # suggest.float -> sample from a continuos space (float)
  # "gamma" - name (to be showcased in the final plot)
  # log - means sample from log space
  gamma = 1 - trial.suggest_float("one_minus_gamma", 0.0001, 0.1, log = True)

  # Create another attribute to store the actual gamma value
  trial.set_user_attr("gamma", gamma)

  ######################################
  # Maximum value for gradient clipping #
  ######################################
  max_grad_norm = trial.suggest_float("max_grad_norm", 0.3, 0.5, log=True)

  ############################################################
  # Number of steps to run for each environment per update #
  ############################################################
  n_steps = 2 ** trial.suggest_int("exponent_n_steps", 3, 10)

  # Create another attribute to store the actual n_steps
  trial.set_user_attr("n_steps", n_steps)

  #################
  # Learning_rate #
  #################
  learning_rate = trial.suggest_float("learning_rate", 1e-5, 1, log=True)

  ########################
  # Network architecture #
  ########################
  # https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html#a2c-policies

  net_arch = trial.suggest_categorical("net_arch", ["tiny", "small"])

  # The net_arch expects a list, that is why it is wrapped in a list
  net_arch = [{"pi": [64], "vf": [64]} if net_arch == 'tiny' \
```

```
            else {"pi" : [64, 64], "vf" : [64, 64]}]

  ########################
  # Activation Function #
  ########################
  activation_fn = trial.suggest_categorical("activation_fn", ["tanh", "relu"])

  activation_fn = {"tanh": nn.Tanh, "relu": nn.ReLU}[activation_fn]



  # Note: The key used in this dictionary match the key used in defining the models
  # Therefore, the naming convention must be followed.
  params = {"n_steps": n_steps,
          "gamma": gamma,
          "learning_rate": learning_rate,
          "max_grad_norm": max_grad_norm,
          "policy_kwargs": {"net_arch": net_arch,
                            "activation_fn": activation_fn}}

  return params

```

Step 3: Define objective

- A custom callback function is defined to report the results of periodic evaluations.

```
class TrialEvalCallback(EvalCallback):

  """
  Callback used for evaluating and reporting a trial.

  Args:
    eval_env (gym.env): An evaluation environment.
    trial (Optuna.trial): An Optuna trial object.
    n_eval_episodes (int): Number of evaluation episodes for each evaluation.
    eval_freq (int): Step interval for an intermediate evaluation in each trial.
    deterministic (boolean): Whether the evaluation should use stochastic or deterministic policy.
    verbose (int):

  Returns:
    out (boolean):
  """

  def __init__(self, eval_env, trial, n_eval_episodes, eval_freq, deterministic, verbose = 0):

    super().__init__(eval_env = eval_env, n_eval_episodes = n_eval_episodes,
                     eval_freq = eval_freq, deterministic = deterministic,
                     verbose = verbose)
    self.trial = trial
    self.eval_idx = 0
    self.is_pruned = False

  def _on_step(self):
    if self.eval_freq > 0 and self.n_calls % self.eval_freq == 0:
      super()._on_step()
      self.eval_idx += 1

      # Send report to optuna
      self.trial.report(self.last_mean_reward, self.eval_idx)

      # Prune trial if needed
      if self.trial.should_prune():
        self.is_pruned = True
        return False
    return True
```

- The true objective function.

```
def objective(trial):

  """
  A function that returns the objective score that decides the quality of a set of hyperparameter.

  Args:
    trial (optuna.Trial): An Optuna trial object.

```

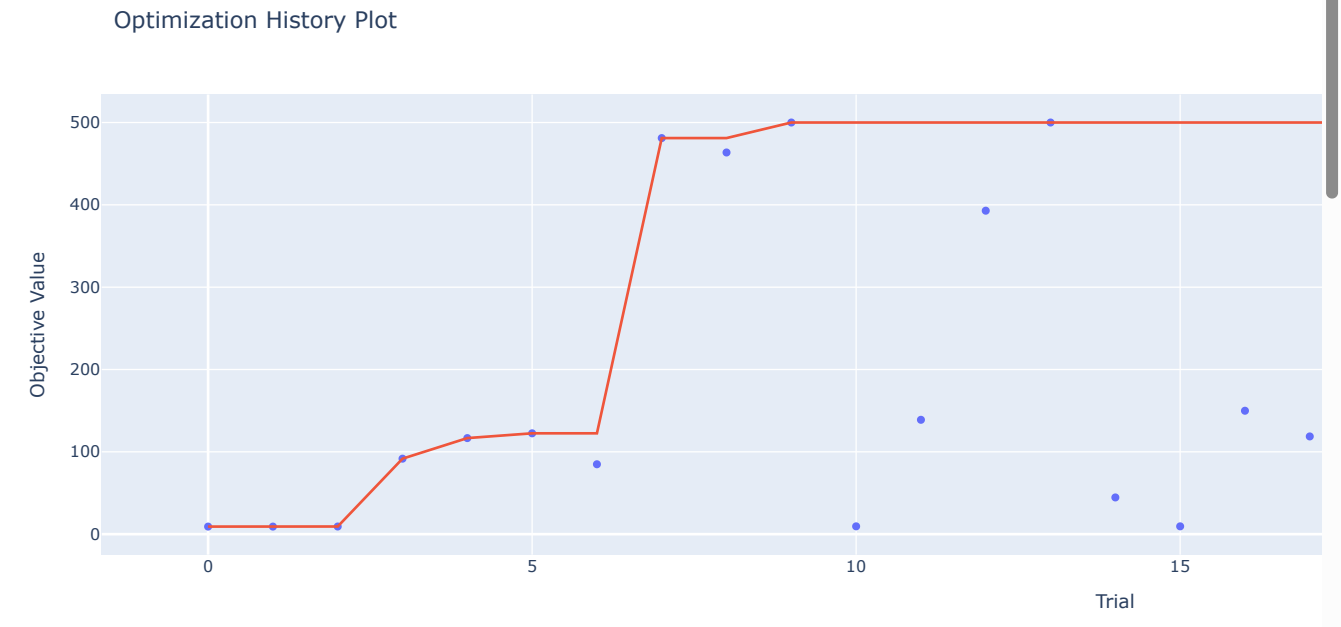
```
  Returns:
    objective_score (float): The score that represents the quality of this set of hyperparameter.
  """

  # Creat the default keyword arguments (those that wasnt defined in the hyperparameter sampling function)
  kwargs = {"policy": "MlpPolicy",
            "env": ENV_ID}

  # Update with the inclusion of the sampled hyperparameter
  kwargs.update(sample_a2c_params(trial))

  # Create a model using the sampled hyperparameter
  model = A2C(**kwargs)

  # Create the environments
  eval_envs = make_vec_env(env_id = ENV_ID,
                           n_envs = N_EVAL_ENVS)

  # Create the call back for reporting evaluation results
  eval_callback = TrialEvalCallback(eval_env = eval_envs,
                                    trial = trial,
                                    n_eval_episodes = N_EVAL_EPISODES,
                                    eval_freq = EVAL_FREQ,
                                    deterministic = True)

  nan_encountered = False
  try:
    model.learn(N_TIMESTEPS, callback = eval_callback)
  except AssertionError as e:
    # Sometimes, randomly sampled error can lead to NaN
    print(e)
    nan_encountered = True
  finally:
    # At the end of training or if error is encountered
    # Free Memory
    model.env.close()
    eval_envs.close()

  # Inform the optimizer that a non-valid hyperparameter is sampled
  if nan_encountered:
    return float('nan')

  if eval_callback.is_pruned:
    raise optuna.exceptions.TrialPruned()

  return eval_callback.last_mean_reward


```

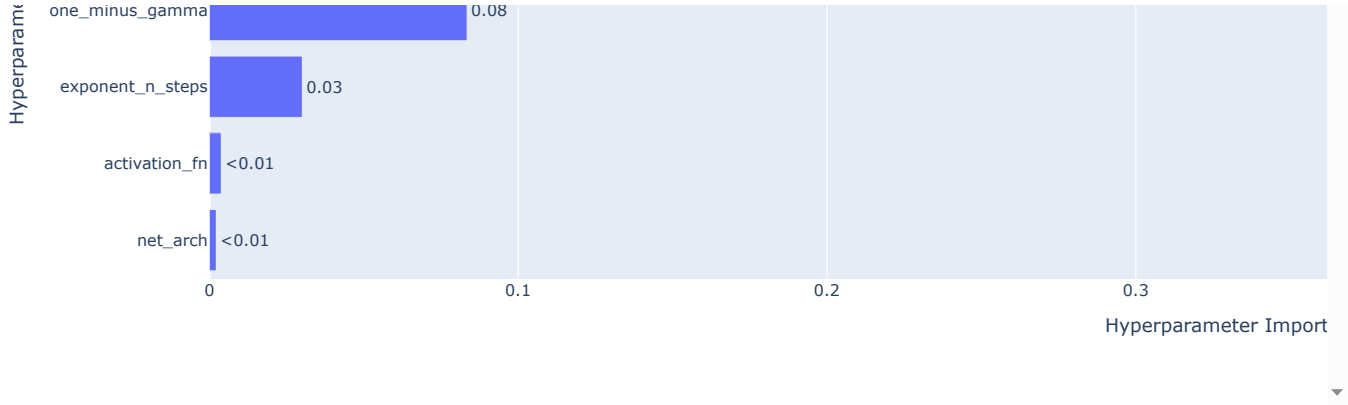
Step 4: Define Hyperparameter Optimization Loop

```
# Set PyTorch num threads to 1 for faster training
# Parallel environement will demand heavy use of CPU.
# Therefore, this line limits to the usage of cpu for PyTorch to be only 1 line.
th.set_num_threads(1)

# Select a sampler
# https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html
# n_startup_trials -> Number of trials at the beginning that sample a set of hyperparameter randomly instead of using the algorithm
# This allows the creation of initial database.
sampler = TPESampler(n_startup_trials = N_STARTUP_TRIALS)

# Select a scheduler / pruner
# https://optuna.readthedocs.io/en/stable/reference/generated/optuna.pruners.MedianPruner.html
# n_startup_trials -> Pruning is disabaled at the beginning for this many trials for initial database creation.
# n_warmup_steps -> Number of steps in each trial that disable the pruning.
pruner = MedianPruner(n_startup_trials = N_STARTUP_TRIALS,
                      n_warmup_steps = N_TIMESTEPS // 3)

# Create a study for Hyperparameter Optimization
# https://optuna.readthedocs.io/en/stable/reference/generated/optuna.create_study.html
study = optuna.create_study(sampler = sampler,
                            pruner = pruner,
                            direction = "maximize")

try:
  # https://optuna.readthedocs.io/en/stable/reference/generated/optuna.study.Study.html#optuna.study.Study.optimize
  study.optimize(objective,
                 n_trials = N_TRIALS,
```

```
                timeout = TIMEOUT,
                n_jobs = N_JOBS)
except KeyboardInterrupt:
  pass

# Print the meta info for the hyperparameter optimization process
print(f"Number of finished trials: {len(study.trials)}")
trial = study.best_trial
print(f"Best trial: {trial.value}")

print("Params: ")
for key, value in trial.params.items():
  print(f"  {key}: {value}")

print("User Attributes: ")
for key, value in trial.user_attrs.items():
  print(f"  {key}: {value}")


# Write report
study.trials_dataframe().to_csv(f"study_result_{ALGO_NAME}_{ENV_ID}.csv")

# Show plot
fig1 = plot_optimization_history(study)
fig2 = plot_param_importances(study)

fig1.show()
fig2.show()

```

```
[I 2025-06-12 09:35:47,739] A new study created in memory with name: no-name-813ee568-ff50-443a-8067-43acc1197b30
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/policies.py:460: UserWarning:

As shared layers in the mlp_extractor are removed since SB3 v1.8.0, you should now pass directly a dictionary and not a list (net_

[I 2025-06-12 09:36:19,525] Trial 0 finished with value: 9.2 and parameters: {'one_minus_gamma': 0.012924707618275728, 'max_grad_n
[I 2025-06-12 09:36:49,605] Trial 1 finished with value: 9.2 and parameters: {'one_minus_gamma': 0.038723746654246494, 'max_grad_n
[I 2025-06-12 09:37:18,442] Trial 2 finished with value: 9.3 and parameters: {'one_minus_gamma': 0.002021039321497278, 'max_grad_n
[I 2025-06-12 09:37:45,863] Trial 3 finished with value: 91.7 and parameters: {'one_minus_gamma': 0.0008731735219335531, 'max_grad
[I 2025-06-12 09:38:14,860] Trial 4 finished with value: 116.7 and parameters: {'one_minus_gamma': 0.0012304139554903581, 'max_gra
[I 2025-06-12 09:38:43,782] Trial 5 finished with value: 122.5 and parameters: {'one_minus_gamma': 0.0001086993609073862, 'max_gra
[I 2025-06-12 09:39:24,908] Trial 6 finished with value: 84.9 and parameters: {'one_minus_gamma': 0.000103139578231042, 'max_grad_
[I 2025-06-12 09:39:54,671] Trial 7 finished with value: 481.1 and parameters: {'one_minus_gamma': 0.0001009578329639844 5, 'max_gr
[I 2025-06-12 09:40:23,040] Trial 8 finished with value: 463.6 and parameters: {'one_minus_gamma': 0.0003560617767207212 3, 'max_gr
[I 2025-06-12 09:40:55,476] Trial 9 finished with value: 500.0 and parameters: {'one_minus_gamma': 0.008262752774391171, 'max_grad
[I 2025-06-12 09:41:33,076] Trial 10 finished with value: 9.5 and parameters: {'one_minus_gamma': 0.007654797065370172, 'max_grad_
[I 2025-06-12 09:42:04,835] Trial 11 finished with value: 138.9 and parameters: {'one_minus_gamma': 0.08561534553697434, 'max_grad
[I 2025-06-12 09:42:35,400] Trial 12 finished with value: 392.9 and parameters: {'one_minus_gamma': 0.007621459534012113, 'max_gra
[I 2025-06-12 09:43:05,790] Trial 13 finished with value: 500.0 and parameters: {'one_minus_gamma': 0.0004338676441916004, 'max_gr
[I 2025-06-12 09:43:39,324] Trial 14 finished with value: 44.6 and parameters: {'one_minus_gamma': 0.0004291344390293505 4, 'max_gr
[I 2025-06-12 09:44:08,172] Trial 15 finished with value: 9.5 and parameters: {'one_minus_gamma': 0.003222336477326619, 'max_grad_
[I 2025-06-12 09:44:41,767] Trial 16 finished with value: 149.9 and parameters: {'one_minus_gamma': 0.004097294434259803, 'max_gra
[I 2025-06-12 09:45:10,447] Trial 17 finished with value: 118.7 and parameters: {'one_minus_gamma': 0.023388281427777047, 'max_gra
[I 2025-06-12 09:45:42,328] Trial 18 finished with value: 266.5 and parameters: {'one_minus_gamma': 0.00039643173176913117, 'max_g
[I 2025-06-12 09:46:14,635] Trial 19 finished with value: 208.1 and parameters: {'one_minus_gamma': 0.0011274364044843253, 'max_gr
[I 2025-06-12 09:46:49,696] Trial 20 finished with value: 110.1 and parameters: {'one_minus_gamma': 0.0002099399167953067, 'max_gr
[I 2025-06-12 09:47:21,125] Trial 21 finished with value: 294.8 and parameters: {'one_minus_gamma': 0.0001840475379181496, 'max_gr
[I 2025-06-12 09:47:55,061] Trial 22 finished with value: 428.9 and parameters: {'one_minus_gamma': 0.0006772044746225003, 'max_gr
[I 2025-06-12 09:48:27,116] Trial 23 finished with value: 172.5 and parameters: {'one_minus_gamma': 0.0002309687834281705, 'max_gr
[I 2025-06-12 09:49:01,142] Trial 24 finished with value: 406.1 and parameters: {'one_minus_gamma': 0.005463870698583285, 'max_gra
[I 2025-06-12 09:49:32,196] Trial 25 finished with value: 500.0 and parameters: {'one_minus_gamma': 0.0015837290597985752, 'max_gr
[I 2025-06-12 09:50:01,660] Trial 26 finished with value: 9.1 and parameters: {'one_minus_gamma': 0.002124909567287219, 'max_grad_
[I 2025-06-12 09:50:40,218] Trial 27 finished with value: 65.6 and parameters: {'one_minus_gamma': 0.001853162648692653, 'max_grad
[I 2025-06-12 09:51:11,973] Trial 28 finished with value: 9.2 and parameters: {'one_minus_gamma': 0.016521266890261135, 'max_grad_
Number of finished trials: 29
Best trial: 500.0
Params:
  one_minus_gamma: 0.008262752774391171
  max_grad_norm: 0.4988993250029
  exponent_n_steps: 5
  learning_rate: 0.0009036800602866176
  net_arch: small
  activation_fn: tanh
User Attributes:
  gamma: 0.9917372472256089
  n_steps: 32
```





Practice: Tune a PPO agent that plays LunarLander-v2

Step 1: Create a config

```
                                        ############################
                                        # Step 1: Create a config #
                                        ############################

# Hyperparameter Optimization Loop
N_TRIALS = 100 # Maximum number of trials during Hyperparameter Optimization Loop
N_JOBS = 1 # Number of parallel jobs to run during each trials in Hyperparameter Optimization Loop
N_STARTUP_TRIALS = 5 # Number of trials to perform random sampling (without relying on sampler) during the Hypeparameter Optimizatic
TIMEOUT = int(60*15) # Maximum number of times (in seconds) the entire loop is allowed up to.

# Evaluation Parameter for each set of hyperparameter
N_TIMESTEPS = int(1e4) # Training budget - Number of time steps in one FULL TRIAL for each set of hyperparameter.
N_EVALUATIONS = 2 # Number of intermediate evaluations performed in one FULL TRIAL for each set of hyperparameter.
EVAL_FREQ = int(N_TIMESTEPS / N_EVALUATIONS) # Step interval for each intermediate evaluations during one FULL TRIAL.
N_EVAL_EPISODES = 10 # Number of episodes to be sampled for each evaluation.


# Environment Parameter
N_EVAL_ENVS = 16 # Number of environments used in parallel during evaluation.

ENV_ID = "LunarLander-v2" # ID of the environments, to be initiated with gym.make()

# Algorithm Parameter
ALGO_NAME = "PPO"
```

Step 2: Define the search space

```
                            ################################################################
                            # Step 2: Define a function that samples the hyperparameter #
                            ################################################################

def sample_ppo_params(trial):

  """
  Sample a set of hyperparameter to be trial'd.

  Args:
    trial (optuna.Trial) : An Optuna trial object.

  Returns:
    params (dict): The hyperparameters to be trial'd. Its key matches the keywords used in defining models.
  """

  # To study the hyperparameter to be updated.

  ###################
  # Discount factor #
  ###################
  # suggest.float -> sample from a continuos space (float)
  # "gamma" - name (to be showcased in the final plot)
  # log - means sample from log space
  gamma = 1 - trial.suggest_float("one_minus_gamma", 0.0001, 0.1, log = True)
```

```

    # Create another attribute to store the actual gamma value
    trial.set_user_attr("gamma", gamma)

    ######################################
    # Maximum value for gradient clipping #
    ######################################
    max_grad_norm = trial.suggest_float("max_grad_norm", 0.3, 0.5, log=True)

    ##########################################################
    # Number of steps to run for each environment per update #
    ##########################################################
    n_steps = 2 ** trial.suggest_int("exponent_n_steps", 3, 11)

    # Create another attribute to store the actual n_steps
    trial.set_user_attr("n_steps", n_steps)

    #################
    # Learning_rate #
    #################
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1, log=True)

    ########################
    # Network architecture #
    ########################
    net_arch = trial.suggest_categorical("net_arch", ["tiny", "small"])

    # The net_arch expects a list, that is why it is wrapped in a list
    net_arch = [{"pi": [64], "vf": [64]} if net_arch == 'tiny' \
                else {"pi" : [64, 64], "vf" : [64, 64]}]

    ########################
    # Activation Function #
    ########################
    activation_fn = trial.suggest_categorical("activation_fn", ["tanh", "relu"])

    activation_fn = {"tanh": nn.Tanh, "relu": nn.ReLU}[activation_fn]



    # Note: The key used in this dictionary match the key used in defining the models
    # Therefore, the naming convention must be followed.
    params = {"n_steps": n_steps,
              "gamma": gamma,
              "learning_rate": learning_rate,
              "max_grad_norm": max_grad_norm,
              "policy_kwargs": {"net_arch": net_arch,
                                "activation_fn": activation_fn}}

    return params
```

Step 3: Define objective functions

```


                                ##############################
                                # Step 3: Objective function #
                                ##############################
                                # 3.1: Callback
class TrialEvalCallback(EvalCallback):

  """
  Callback used for evaluating and reporting a trial.

  Args:
    eval_env (gym.env): An evaluation environment.
    trial (Optuna.trial): An Optuna trial object.
    n_eval_episodes (int): Number of evaluation episodes for each evaluation.
    eval_freq (int): Step interval for an intermediate evaluation in each trial.
    deterministic (boolean): Whether the evaluation should use stochastic or deterministic policy.
    verbose (int):

  Returns:
    out (boolean):
  """

  def __init__(self, eval_env, trial, n_eval_episodes, eval_freq, deterministic, verbose = 0):

    super().__init__(eval_env = eval_env, n_eval_episodes = n_eval_episodes,
                     eval_freq = eval_freq, deterministic = deterministic,
```

```
                    verbose = verbose)
    self.trial = trial
    self.eval_idx = 0
    self.is_pruned = False

  def _on_step(self):
    if self.eval_freq > 0 and self.n_calls % self.eval_freq == 0:
      super()._on_step()
      self.eval_idx += 1

      # Send report to optuna
      self.trial.report(self.last_mean_reward, self.eval_idx)

      # Prune trial if needed
      if self.trial.should_prune():
        self.is_pruned = True
        return False
    return True

                            # 3.2: The definition of the objective score.
def objective(trial):

  """
  A function that returns the objective score that decides the quality of a set of hyperparameter.

  Args:
    trial (optuna.Trial): An Optuna trial object.

  Returns:
    objective_score (float): The score that represents the quality of this set of hyperparameter.
  """

  # Creat the default keyword arguments (those that wasnt defined in the hyperparameter sampling function)
  kwargs = {"policy": "MlpPolicy",
            "env": ENV_ID}

  # Update with the inclusion of the sampled hyperparameter
  kwargs.update(sample_ppo_params(trial))

  # Create a model using the sampled hyperparameter
  model = PPO(**kwargs)

  # Create the environments
  eval_envs = make_vec_env(env_id = ENV_ID,
                           n_envs = N_EVAL_ENVS)

  # Create the call back for reporting evaluation results
  eval_callback = TrialEvalCallback(eval_env = eval_envs,
                                    trial = trial,
                                    n_eval_episodes = N_EVAL_EPISODES,
                                    eval_freq = EVAL_FREQ,
                                    deterministic = True)

  nan_encountered = False
  try:
    model.learn(N_TIMESTEPS, callback = eval_callback)
  except AssertionError as e:
    # Sometimes, randomly sampled error can lead to NaN
    print(e)
    nan_encountered = True
  finally:
    # At the end of training or if error is encountered
    # Free Memory
    model.env.close()
    eval_envs.close()

  # Inform the optimizer that a non-valid hyperparameter is sampled
  if nan_encountered:
    return float('nan')

  if eval_callback.is_pruned:
    raise optuna.exceptions.TrialPruned()

  return eval_callback.last_mean_reward
```

Step 4: Hyperparameter Optimization Loop

```

                            ##############################
                            # Step 4: Optimization Loop #
```

```
                            ############################

# Set PyTorch num threads to 1 for faster training
# Parallel environement will demand heavy use of CPU.
# Therefore, this line limits to the usage of cpu for PyTorch to be only 1 line.
th.set_num_threads(1)

# Select a sampler
# https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html
# n_startup_trials -> Number of trials at the beginning that sample a set of hyperparameter randomly instead of using the algorithm
# This allows the creation of initial database.
sampler = TPESampler(n_startup_trials = N_STARTUP_TRIALS)

# Select a scheduler / pruner
# https://optuna.readthedocs.io/en/stable/reference/generated/optuna.pruners.MedianPruner.html
# n_startup_trials -> Pruning is disabaled at the beginning for this many trials for initial database creation.
# n_warmup_steps -> Number of steps in each trial that disable the pruning.
pruner = MedianPruner(n_startup_trials = N_STARTUP_TRIALS,
                      n_warmup_steps = N_TIMESTEPS // 3)

# Create a study for Hyperparameter Optimization
# https://optuna.readthedocs.io/en/stable/reference/generated/optuna.create_study.html
study = optuna.create_study(sampler = sampler,
                            pruner = pruner,
                            direction = "maximize")

try:
  # https://optuna.readthedocs.io/en/stable/reference/generated/optuna.study.Study.html#optuna.study.Study.optimize
  study.optimize(objective,
                 n_trials = N_TRIALS,
                 timeout = TIMEOUT,
                 n_jobs = N_JOBS)
except KeyboardInterrupt:
  pass

# Print the meta info for the hyperparameter optimization process
print(f"Number of finished trials: {len(study.trials)}")
trial = study.best_trial
print(f"Best trial: {trial.value}")

print("Params: ")
for key, value in trial.params.items():
  print(f"  {key}: {value}")

print("User Attributes: ")
for key, value in trial.user_attrs.items():
  print(f"  {key}: {value}")


# Write report
study.trials_dataframe().to_csv(f"study_result_{ALGO_NAME}_{ENV_ID}.csv")

# Show plot
fig1 = plot_optimization_history(study)
fig2 = plot_param_importances(study)

fig1.show()
fig2.show()
```

Show hidden output

Practice: Tune a DQN agent that plays Atari Games - Space Invader

Install Libraries

```
!pip install git+https://github.com/DLR-RM/rl-baselines3-zoo
```

Show hidden output

```
!pip install gymnasium[atari]
!pip install gymnasium[accept-rom_license]
```

Show hidden output

Create a config file

- Save the config file as `dqn.yml`
- Note: The config file is only moslty used as a placeholder. During Hyperparameter optimization, a set of hyperparameter will be randomly sampled, thus replacing these values.
- The type and range of hyperparameter values to be sampled may be referred to: https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/rl_zoo3/hyperparams_opt.py#L222

```
# The default config file
SpaceInvadersNoFrameskip-v4:
  env_wrapper:
    - stable_baselines3.common.atari_wrappers.AtariWrapper
  frame_stack: 4 #Every 4 frame as 1 input to allow the model to learn the trajectories of the object.
  policy: 'CnnPolicy'
  n_timesteps: !!float 1e2 # 1e6 (Recommended, but shortened in this notebook as its only for demo)
  buffer_size: 100000
  learning_rate: !!float 1e-4
  batch_size: 32
  learning_starts: 100000
  target_update_interval: 1000
  train_freq: 4
  gradient_steps: 1
  exploration_fraction: 0.1
  exploration_final_eps: 0.01
  # If True, you need to deactivate handle_timeout_termination
  # in the replay_buffer_kwargs
  optimize_memory_usage: False
```

Access the API for hyperparameter optimization (Built-in)

- Refer to the raw code to find out the usage: https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/rl_zoo3/train.py
- The full command is as following: `!python -m rl_zoo3.train --algo dqn --env SpaceInvadersNoFrameskip-v4 -f logs/ -c dqn.yml -optimize --optimization-log-path logs/optimization --eval-episodes 10 --n-eval-envs 1 --max-total-trials 100 --n-jobs 1 --sampler "tpe" --pruner "median" --n-startup-trials 10 --n-evaluations 2`

```
!python -m rl_zoo3.train --algo dqn --env SpaceInvadersNoFrameskip-v4 -f logs/ -c dqn.yml -optimize --optimization-log-path logs/opt
```

Show hidden output

Post Notes: How to apply it onto a PyTorch Model?

- A good reference link: https://www.geeksforgeeks.org/hyperparameter-tuning-with-optuna-in-pytorch/
- The main difference will be on the definition of the objective function and how to manually report back the intermediate and/or final evaluation to optune.
- The following codes are generated by chatGPT on how to accomplish both. (Note: It has not yet to be tested)

```
def objective(trial):
    # Sample hyperparameters
    lr = trial.suggest_float("lr", 1e-5, 1e-1, log=True)
    n_units = trial.suggest_int("n_units", 16, 128)

    # Model
```