Machine Learning for Graphs - BECKER Gonzalo, BIGOIS Gautier

Graph Neural Networking Challenge 2023

Our notebook is based on the Graph Neural Networking Challenge 2023.

This challenge's goal is to create a digital twin of a network, using real network data. A flow is a group of packets. Our aim is to predict the mean packet delay per flow from multiple features that we'll be describing further.

Let's begin by understanding the dataset and trying the baseline given for the challenge

Imports and dataset loading

In [ ]:

```
import tensorflow as tf
from google.colab import drive
from google.colab import files
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import cvxpy as cp
import scipy as scipy
import cvxopt as cvxopt
!pip install cvxpylayers
import cvxpylayers as cvxpylayers
from numpy.linalg import matrix_rank
from cvxpylayers.torch import CvxpyLayer
from sklearn.model_selection import train_test_split
import argparse
import os
import shutil
```

```
Requirement already satisfied: cvxpylayers in /usr/local/lib/python3.10/dist
-packages (0.1.6)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist
-packages (from cvxpylayers) (1.23.5)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dis
t-packages (from cvxpylayers) (1.11.4)
Requirement already satisfied: diffcp>=1.0.13 in /usr/local/lib/python3.10/d
ist-packages (from cvxpylayers) (1.0.23)
Requirement already satisfied: cvxpy>=1.1.0a4 in /usr/local/lib/python3.10/d
ist-packages (from cvxpylayers) (1.3.3)
Requirement already satisfied: osqp>=0.4.1 in /usr/local/lib/python3.10/dist
-packages (from cvxpy>=1.1.0a4->cvxpylayers) (0.6.2.post8)
Requirement already satisfied: ecos>=2 in /usr/local/lib/python3.10/dist-pac
kages (from cvxpy>=1.1.0a4->cvxpylayers) (2.0.12)
Requirement already satisfied: scs>=1.1.6 in /usr/local/lib/python3.10/dist-
packages (from cvxpy>=1.1.0a4->cvxpylayers) (3.2.4.post1)
Requirement already satisfied: setuptools>65.5.1 in /usr/local/lib/python3.1
0/dist-packages (from cvxpy>=1.1.0a4->cvxpylayers) (67.7.2)
Requirement already satisfied: pybind11>=2.4 in /usr/local/lib/python3.10/di
st-packages (from diffcp>=1.0.13->cvxpylayers) (2.11.1)
Requirement already satisfied: threadpoolctl>=1.1 in /usr/local/lib/python3.
10/dist-packages (from diffcp>=1.0.13->cvxpylayers) (3.2.0)
Requirement already satisfied: qdldl in /usr/local/lib/python3.10/dist-packa
ges (from osqp>=0.4.1->cvxpy>=1.1.0a4->cvxpylayers) (0.1.7.post0)
```

For this project, we'll only be using a small part of the dataset, in order to be able to train some models within a short time period (<15 min).

Please, download the Dataset from the drive link and then upload it to the notebook

In [ ]:
```
# Import data_cbr_mb_cv.zip
# https://drive.google.com/file/d/1s4QKBvivMYM2IUHjhbZgagVxdljhCH7Y/view?usp
uploaded = files.upload()
!unzip data_cbr_mb_cv.zip
%ls
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data_cbr_mb_cv.zip to data_cbr_mb_cv.zip
Archive:  data_cbr_mb_cv.zip
   creating: data_cbr_mb_cv/
   creating: data_cbr_mb_cv/0/
   creating: data_cbr_mb_cv/0/training/
   creating: data_cbr_mb_cv/0/training/15428484028351491192/
   creating: data_cbr_mb_cv/0/training/15428484028351491192/00000000.shard/
  inflating: data_cbr_mb_cv/0/training/15428484028351491192/00000000.shard/0
0000000.snapshot
  inflating: data_cbr_mb_cv/0/training/dataset_spec.pb
  inflating: data_cbr_mb_cv/0/training/snapshot.metadata
   creating: data_cbr_mb_cv/0/validation/
   creating: data_cbr_mb_cv/0/validation/9181049271269547031/
   creating: data_cbr_mb_cv/0/validation/9181049271269547031/00000000.shard/
  inflating: data_cbr_mb_cv/0/validation/9181049271269547031/00000000.shard/
00000000.snapshot
  inflating: data_cbr_mb_cv/0/validation/dataset_spec.pb
  inflating: data_cbr_mb_cv/0/validation/snapshot.metadata
   creating: data_cbr_mb_cv/1/
   creating: data_cbr_mb_cv/1/training/
   creating: data_cbr_mb_cv/1/training/9209107630152679508/
   creating: data_cbr_mb_cv/1/training/9209107630152679508/00000000.shard/
  inflating: data_cbr_mb_cv/1/training/9209107630152679508/00000000.shard/00
000000.snapshot
  inflating: data_cbr_mb_cv/1/training/dataset_spec.pb
  inflating: data_cbr_mb_cv/1/training/snapshot.metadata
   creating: data_cbr_mb_cv/1/validation/
   creating: data_cbr_mb_cv/1/validation/2619890786369781981/
   creating: data_cbr_mb_cv/1/validation/2619890786369781981/00000000.shard/
  inflating: data_cbr_mb_cv/1/validation/2619890786369781981/00000000.shard/
00000000.snapshot
  inflating: data_cbr_mb_cv/1/validation/dataset_spec.pb
  inflating: data_cbr_mb_cv/1/validation/snapshot.metadata
   creating: data_cbr_mb_cv/2/
   creating: data_cbr_mb_cv/2/training/
   creating: data_cbr_mb_cv/2/training/9103858788583102920/
   creating: data_cbr_mb_cv/2/training/9103858788583102920/00000000.shard/
  inflating: data_cbr_mb_cv/2/training/9103858788583102920/00000000.shard/00
000000.snapshot
  inflating: data_cbr_mb_cv/2/training/dataset_spec.pb
  inflating: data_cbr_mb_cv/2/training/snapshot.metadata
   creating: data_cbr_mb_cv/2/validation/
   creating: data_cbr_mb_cv/2/validation/9996255799318782 75/
```

```
   creating: data_cbr_mb_cv/3/validation/
   creating: data_cbr_mb_cv/3/validation/15648048243224390985/
   creating: data_cbr_mb_cv/3/validation/15648048243224390985/00000000.shar
d/
  inflating: data_cbr_mb_cv/3/validation/15648048243224390985/00000000.shar
d/00000000.snapshot
  inflating: data_cbr_mb_cv/3/validation/dataset_spec.pb
  inflating: data_cbr_mb_cv/3/validation/snapshot.metadata
   creating: data_cbr_mb_cv/4/
   creating: data_cbr_mb_cv/4/training/
   creating: data_cbr_mb_cv/4/training/2710535624285927582/
   creating: data_cbr_mb_cv/4/training/2710535624285927582/00000000.shard/
  inflating: data_cbr_mb_cv/4/training/2710535624285927582/00000000.shard/00
000000.snapshot
  inflating: data_cbr_mb_cv/4/training/dataset_spec.pb
  inflating: data_cbr_mb_cv/4/training/snapshot.metadata
   creating: data_cbr_mb_cv/4/validation/
   creating: data_cbr_mb_cv/4/validation/17645047073329822252/
   creating: data_cbr_mb_cv/4/validation/17645047073329822252/00000000.shar
d/
  inflating: data_cbr_mb_cv/4/validation/17645047073329822252/00000000.shar
d/00000000.snapshot
  inflating: data_cbr_mb_cv/4/validation/dataset_spec.pb
  inflating: data_cbr_mb_cv/4/validation/snapshot.metadata
data_cbr_mb_cv/  data_cbr_mb_cv.zip  sample_data/
```

Let's have a look on the data :

In [ ]:
```
# load the training dataset
ds_train = tf.data.Dataset.load(f"data_cbr_mb_cv/0/training", compression="G
columns = ds_train.element_spec
print(columns)
```

```
({'path_to_link': RaggedTensorSpec(TensorShape([None, None, 2]), tf.int32,
1, tf.int64), 'flow_packet_size': TensorSpec(shape=(None, 1), dtype=tf.float
32, name=None), 'flow_packets': TensorSpec(shape=(None, 1), dtype=tf.float3
2, name=None), 'sample_file_id': TensorSpec(shape=(None,), dtype=tf.int32, n
ame=None), 'flow_id': TensorSpec(shape=(None,), dtype=tf.string, name=None),
'sample_file_name': TensorSpec(shape=(None,), dtype=tf.string, name=None),
'link_capacity': TensorSpec(shape=(None, 1), dtype=tf.float32, name=None),
'flow_traffic': TensorSpec(shape=(None, 1), dtype=tf.float32, name=None), 'l
ink_to_path': RaggedTensorSpec(TensorShape([None, None]), tf.int32, 1, tf.in
t64), 'flow_type': TensorSpec(shape=(None, 2), dtype=tf.float32, name=None),
'flow_length': TensorSpec(shape=(None,), dtype=tf.int32, name=None)}, Tensor
Spec(shape=<unknown>, dtype=tf.float32, name=None))
```

We have the following features :

```
flow_traffic: the average traffic bandwidth per flow in bps
flow_packets: the number of generated packets per flow
flow_packet_size: the size of the generated packets per flow
flow_type: two-dimensional one-hot encoded feature used to
identify the flow type of each flow
    [1, 0] indicates the flow is a Constant Bit Rate (CBR)
flow
```

```
    [0, 1] indicates the flow is a Multi Burst (MB) flow
flow_length: length of the physical path followed by each
flow
link_capacity: for each link, it indicates its bandwidth in
bps
link_to_path: for each flow, it indicates the links forming
its path, in order
path_to_link: for each link, it lists the flows that traverse
it. It also includes the position of the link in each flow's
path. For a given link the same flow can appear more than
once if the link is traversed more than one in the same flow
path
```

It's interesting to note that we do not have a direct access to the nodes of the graph themselves but to the paths and the links of the graph

Algorithm

A detailed explanation of the implemented algorithm will be given in this section.

As a whole, the algorithm defines a differenciable function where the input is given by the different flows in the network and their corresponding paths given by a sequence of links, and the output is given by a QoS metric, in particular the average delay per flow in the network. Then, a training can be performed so as to find the parameters that will better estimate the delay for an aribitrary network topogy.

Let $\{f_i\}$ be the flows, $\{q_i\}$ the queues on each device and $\{l_i\}$ the links. Let $\boldsymbol{h}$ denote some embedded features for each flow, queue or link. Let then G_f, G_q and G_l be functions representing the complex relationships between the different elements of the network. Then, the problem to be solved can be stated as:

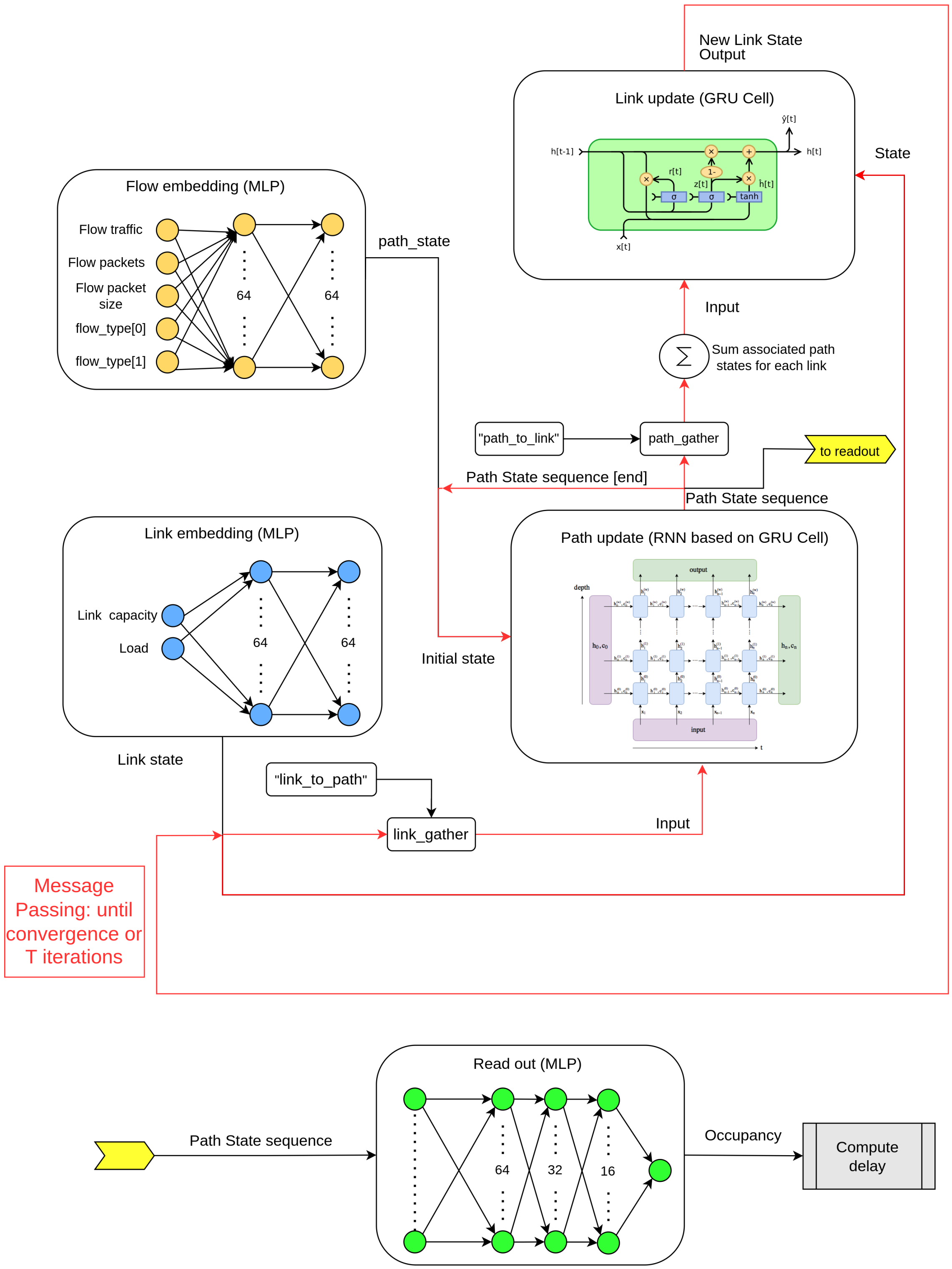
$$\boldsymbol{h_{f_i}} = G_f(\boldsymbol{h}_q^{i,1}, \boldsymbol{h}_l^{i,1}, \ldots, \boldsymbol{h}_q^{i,M}, \boldsymbol{h}_l^{i,M})$$

$$\boldsymbol{h_{q_j}} = G_q(\boldsymbol{h}_{f_1}, \ldots, \boldsymbol{h}_{f_I})$$

$$\boldsymbol{h_{l_k}} = G_l(\boldsymbol{h}_{q_1}, \ldots, \boldsymbol{h}_{q_J})$$

There are some clear circular dependencies, since links and queues define flow behavior, while flows at the same time affect queues and therefore links themselves. This is the reason why a simple Message Passing Neural Network (MPNN) where nodes are taken as the different routing devices would not work. Indeed, in a message passing scheme where only nodes (devices) communicate, the different routing paths for each flow would not be considered. That is, given a path as a sequence of links, the information about how one link affects the other (because of capacity limitations) is not retained by such a model.

The proposed model (https://arxiv.org/pdf/2310.11889.pdf arises then as an alternative to correctly treat this complex interrelationships. In this algorithm, the message passing stage does not only take place between nodes, but also between links. This algorithm can be described by the following flux diagram:



The flow and link embeddings are obtained through a Multi-Layer Perceptron (MLP) and the input is given by some selected features. Note that flow_type is given two features to select wether the device is a switch or a router.

Once this intial flow and link embedding states are computed, a Message Passing stage takes place. In each loop, the "link_to_path" variable is used to choose the links for each path, and then assign to each path the corresponding link current state. This sequence is then sent to a Recurrent Neural Network (RNN) based on Gated Recurrent Unit (GRU) Cells. GRUs are a gating mechanism that smartly remembers or forgets features so as to retain complex and long-lasting dependencies in sequential data. The whole internal state sequence is given as output. The final internal state is then redefined as the new flow embedding.

Regarding links, for all paths passing through each link, flow embedding state sequences are summed up. The variable "path_to_link" is used for this purpose. This is the input for a GRU cell, with the previous link state defined as the state input. The output is then the new link state.

Note that the different structures defined for the path update and link update are actually implementing the message passing.

Baseline Model code

The following code is directly extracted from the Baseline Model, we're gonna use it to train several models. You may just collapse the cells and execute the whole 4 cells

In [ ]:
```
"""
   Copyright 2023 Universitat Politècnica de Catalunya

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
"""

import tensorflow as tf

class Baseline_cbr_mb(tf.keras.Model):
    min_max_scores_fields = {
        "flow_traffic",
        "flow_packets",
        "flow_packet_size",
        "link_capacity",
    }
    min_max_scores = None
```

```
    name = "Baseline_cbr_mb"

    def __init__(self, override_min_max_scores=None, name=None):
        super(Baseline_cbr_mb, self).__init__()

        self.iterations = 8
        self.path_state_dim = 64
        self.link_state_dim = 64

        if override_min_max_scores is not None:
            self.set_min_max_scores(override_min_max_scores)
        if name is not None:
            assert type(name) == str, "name must be a string"
            self.name = name

        # GRU Cells used in the Message Passing step
        self.path_update = tf.keras.layers.RNN(
            tf.keras.layers.GRUCell(self.path_state_dim, name="PathUpdate"),
            return_sequences=True,
            return_state=True,
            name="PathUpdateRNN",
        )
        self.link_update = tf.keras.layers.GRUCell(
            self.link_state_dim, name="LinkUpdate"
        )

        self.flow_embedding = tf.keras.Sequential(
            [
                tf.keras.layers.Input(shape=5),
                tf.keras.layers.Dense(
                    self.path_state_dim, activation=tf.keras.activations.rel
                ),
                tf.keras.layers.Dense(
                    self.path_state_dim, activation=tf.keras.activations.rel
                ),
            ],
            name="PathEmbedding",
        )

        self.link_embedding = tf.keras.Sequential(
            [
                tf.keras.layers.Input(shape=2),
                tf.keras.layers.Dense(
                    self.link_state_dim, activation=tf.keras.activations.rel
                ),
                tf.keras.layers.Dense(
                    self.link_state_dim, activation=tf.keras.activations.rel
                ),
            ],
            name="LinkEmbedding",
        )

        self.readout_path = tf.keras.Sequential(
            [
                tf.keras.layers.Input(shape=(None, self.path_state_dim)),
                tf.keras.layers.Dense(
```

```
                self.link_state_dim // 2, activation=tf.keras.activatior
            ),
            tf.keras.layers.Dense(
                self.link_state_dim // 4, activation=tf.keras.activatior
            ),
            tf.keras.layers.Dense(1, activation=tf.keras.activations.sof
        ],
        name="PathReadout",
    )

def set_min_max_scores(self, override_min_max_scores):
    assert (
        type(override_min_max_scores) == dict
        and all(kk in override_min_max_scores for kk in self.min_max_scc
        and all(len(val) == 2 for val in override_min_max_scores.values(
    ), "overriden min-max dict is not valid!"
    self.min_max_scores = override_min_max_scores

@tf.function
def call(self, inputs):
    # Ensure that the min-max scores are set
    assert self.min_max_scores is not None, "the model cannot be called

    # Process raw inputs
    flow_traffic = inputs["flow_traffic"]
    flow_packets = inputs["flow_packets"]
    flow_packet_size = inputs["flow_packet_size"]
    flow_type = inputs["flow_type"]
    link_capacity = inputs["link_capacity"]
    link_to_path = inputs["link_to_path"]
    path_to_link = inputs["path_to_link"]

    path_gather_traffic = tf.gather(flow_traffic, path_to_link[:, :, 0])
    load = tf.math.reduce_sum(path_gather_traffic, axis=1) / (link_capac

    # Initialize the initial hidden state for paths
    path_state = self.flow_embedding(
        tf.concat(
            [
                (flow_traffic - self.min_max_scores["flow_traffic"][0])
                * self.min_max_scores["flow_traffic"][1],
                (flow_packets - self.min_max_scores["flow_packets"][0])
                * self.min_max_scores["flow_packets"][1],
                (flow_packet_size - self.min_max_scores["flow_packet_siz
                * self.min_max_scores["flow_packet_size"][1],
                flow_type,
            ],
            axis=1,
        )
    )


    # Initialize the initial hidden state for links
    link_state = self.link_embedding(
        tf.concat(
            [
```

```
                    (link_capacity - self.min_max_scores["link_capacity"][0]
                    * self.min_max_scores["link_capacity"][1],
                    load,
                ],
                axis=1,
            ),
        )

        # Iterate t times doing the message passing
        for _ in range(self.iterations):
            ####################
            #  LINKS TO PATH   #
            ####################
            link_gather = tf.gather(link_state, link_to_path, name="LinkToPa
            previous_path_state = path_state
            path_state_sequence, path_state = self.path_update(
                link_gather, initial_state=path_state
            )
            # We select the element in path_state_sequence so that it corres
            path_state_sequence = tf.concat(
                [tf.expand_dims(previous_path_state, 1), path_state_sequence
            )

            ###################
            #   PATH TO LINK  #
            ###################
            path_gather = tf.gather_nd(
                path_state_sequence, path_to_link, name="PathToRLink"
            )
            path_sum = tf.math.reduce_sum(path_gather, axis=1)
            link_state, _ = self.link_update(path_sum, states=link_state)

        ################
        #  READOUT     #
        ################

        occupancy = self.readout_path(path_state_sequence[:, 1:])
        capacity_gather = tf.gather(link_capacity, link_to_path)
        delay_sequence = occupancy / capacity_gather
        delay = tf.math.reduce_sum(delay_sequence, axis=1)
        return delay
```

In [ ]:

```python
"""

import os

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import tensorflow as tf
import random
import numpy as np
from typing import List, Optional, Union, Tuple, Dict, Any

# Run eagerly-> Turn true for debugging only
RUN_EAGERLY = False
tf.config.run_functions_eagerly(RUN_EAGERLY)


def _reset_seeds(seed: int = 42) -> None:
    """Reset rng seeds, and also indicate tf if to run eagerly or not

    Parameters
    ----------
    seed : int, optional
        Seed for rngs, by default 42
    """
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)


def get_default_callbacks() -> List[tf.keras.callbacks.Callback]:
    """Returns the default callbacks for the training of the models
    (EarlyStopping and ReduceLROnPlateau callbacks)
    """
    return [
        tf.keras.callbacks.EarlyStopping(
            monitor="val_loss",
            patience=10,
            restore_best_weights=True,
            min_delta=0.0001,
            start_from_epoch=4,
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            factor=0.5,
            patience=5,
            verbose=1,
            mode="min",
            min_delta=0.001,
        ),
    ]


def get_default_hyperparams() -> Dict[str, Any]:
    """Returns the default hyperparameters for the training of the models. T
    - Adam optimizer with lr=0.001
    - MeanAbsolutePercentageError loss
    - No additional metrics
```

```
    - EarlyStopping and ReduceLROnPlateau callbacks
    - 100 epochs
    """
    return {
        "optimizer": tf.keras.optimizers.Adam(learning_rate=0.001),
        "loss": tf.keras.losses.MeanAbsolutePercentageError(),
        "metrics": [],
        "additional_callbacks": get_default_callbacks(),
        "epochs": 3,
    }


def get_min_max_dict(
    ds: tf.data.Dataset, params: List[str], include_y: Optional[str] = None
) -> Dict[str, Tuple[np.ndarray, np.ndarray]]:
    """Get the min and the max-min for different parameters of a dataset. La

    Parameters
    ----------
    ds : tf.data.Dataset
        Training dataset where to base the min-max normalization from.

    params : List[str]
        List of strings indicating the parameters to extract the features fr

    include_y : Optional[str], optional
        Indicates if to also extract the features of the output variable.
        Inputs indicate the string key used on the return dict. If None, it
        By default None.

    Returns
    -------
    Dict[str, Tuple[np.ndarray, np.ndarray]]
        Dictionary containing the values needed for the min-max normalizatio
        The first value is the min value of the parameter, and the second is
    """

    # Use first sample to get the shape of the tensors
    iter_ds = iter(ds)
    sample, label = next(iter_ds)
    params_lists = {param: sample[param].numpy() for param in params}
    if include_y:
        params_lists[include_y] = label.numpy()

    # Include the rest of the samples
    for sample, label in iter_ds:
        for param in params:
            params_lists[param] = np.concatenate(
                (params_lists[param], sample[param].numpy()), axis=0
            )
        if include_y:
            params_lists[include_y] = np.concatenate(
                (params_lists[include_y], label.numpy()), axis=0
            )

    scores = dict()
```

```python
    for param, param_list in params_lists.items():
        min_val = np.min(param_list, axis=0)
        min_max_val = np.max(param_list, axis=0) - min_val
        if min_max_val.size == 1 and min_max_val == 0:
            scores[param] = [min_val, 0]
            print(f"Min-max normalization Warning: {param} has a max-min of
        elif min_max_val.size > 1 and np.any(min_max_val == 0):
            min_max_val[min_max_val != 0] = 1 / min_max_val[min_max_val != 0
            scores[param] = [min_val, min_max_val]
            print(
                f"Min-max normalization Warning: Several values of {param} h
            )
        else:
            scores[param] = [min_val, 1 / min_max_val]

    return scores


def train_and_evaluate(
    ds_path: Union[str, Tuple[str, str]],
    model: tf.keras.Model,
    optimizer: tf.keras.optimizers.Optimizer,
    loss: tf.keras.losses.Loss,
    metrics: List[tf.keras.metrics.Metric],
    additional_callbacks: List[tf.keras.callbacks.Callback],
    epochs: int = 3,
    ckpt_path: Optional[str] = None,
    tensorboard_path: Optional[str] = None,
    restore_ckpt: bool = False,
    final_eval: bool = True,
) -> Tuple[tf.keras.Model, Union[float, np.ndarray, None]]:
    """
    Train the given model with the given dataset, using the provided paramet
    Besides for defining the hyperparameters, refer to get_default_hyperpara

    Parameters
    ----------
    ds_path : str
        Path to the dataset. Datasets are expected to be in tf.data.Dataset
        If ds_path is a string, then it used as the path to both the trainin
        If so, it is expected that the training and validation datasets are
        If ds_path is a tuple of two strings, then the first string is used
        and the second string is used as the path to the validation dataset.

    model : tf.keras.Model
        Instance of the model to train. Besides being a tf.keras.Model, it s
        as the models in the models module.

    optimizer : tf.keras.Optimizer
        Optimizer used by the training process

    loss : tf.keras.losses.Loss
        Loss function to be used by the process

    metrics : List[tf.keras.metrics.Metric]
        List of additional metrics to consider during training
```

```python
    additional_callbacks : List[tf.keras.callbacks.Callback], optional
        List containing tensorflow callback functions to be added to the tra
        A callback to generate tensorboard and checkpoint files at each epoc

    epochs : int, optional
        Number of epochs of in the training process, by default 100

    ckpt_path : Optional[str], optional
        Path where to store the training checkpints, by default "{repository

    tensorboard_path : Optional[str], optional
        Path where to store tensorboard logs, by default "{repository root}/

    restore_ckpt : bool, optional
        If True, before training the model, it is checked if there is a chec
        If so, the model loads the latest checkpoint and continues training

    final_eval : bool, optional
        If True, the model is evaluated on the validation dataset one last t

    Returns
    -------
    Tuple[tf.keras.Model, Union[float, np.ndarray, None]]
        Instance of the trained model, and the result of its evaluation
    """

    # Reset tf state
    _reset_seeds()
    # Check epoch number is valid
    assert epochs > 0, "Epochs must be greater than 0"
    # Load ds
    if isinstance(ds_path, str):
        ds_train = tf.data.Dataset.load(f"{ds_path}/training", compression="
        ds_val = tf.data.Dataset.load(f"{ds_path}/validation", compression="
    else:
        ds_train = tf.data.Dataset.load(ds_path[0], compression="GZIP")
        ds_val = tf.data.Dataset.load(ds_path[1], compression="GZIP")
    # Checkpoint path
    if ckpt_path is None:
        ckpt_path = f"ckpt/{model.name}"
    # Tensorboard path
    if tensorboard_path is None:
        tensorboard_path = f"tensorboard/{model.name}"

    # Apply min-max normalization
    model.set_min_max_scores(get_min_max_dict(ds_train, model.min_max_scores

    # Compile model
    model.compile(
        optimizer=optimizer,
        loss=loss,
        metrics=metrics,
        run_eagerly=RUN_EAGERLY,
    )
    # Load checkpoint
```

```
    if restore_ckpt:
        ckpt = tf.train.latest_checkpoint(ckpt_path)
        if ckpt is not None:
            print("Restoring from checkpoint")
            model.load_weights(ckpt)
        else:
            print(
                f"WARNING: No checkpoint was found at '{ckpt_path}', trainin
            )
    else:
        print("restore_ckpt = False, training from scratch")

    # Create callbacks
    cpkt_callback = tf.keras.callbacks.ModelCheckpoint(
        filepath=os.path.join(ckpt_path, "{epoch:02d}-{val_loss:.4f}"),
        verbose=1,
        mode="min",
        save_best_only=False,
        save_weights_only=True,
        save_freq="epoch",
    )
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=tensorboard_path, histogram_freq=1
    )

    # Train model
    model.fit(
        ds_train,
        validation_data=ds_val,
        epochs=epochs,
        callbacks=[cpkt_callback, tensorboard_callback] + additional_callbac
        use_multiprocessing=True,
    )

    if final_eval:
        return model, model.evaluate(ds_val)
    else:
        return model, None
```

Training

Now let's try to train the model with our dataset, for now we'll just do 3 epochs, just to check that everything is working fine

In [ ]:

```
ds_path = "data_cbr_mb_cv"
model = Baseline_cbr_mb
cfv = False
n_folds = 5

# code for simple training/validation
if not cfv:
    _reset_seeds()
    trained_model, evaluation = train_and_evaluate(
        os.path.join(ds_path, "0"), model(), **get_default_hyperparams()
```

```
    )
    print("Final evaluation:", evaluation)

# code for cross-fold validation
else:
    trained_models = []
    trained_models_val_loss = []
    ckpt_path = f"ckpt/{model.name}_cv/"
    tensorboard_path = f"tensorboard/{model.name}_cv/"

    # Execute each fold
    for fold_idx in range(n_folds):
        print("***** Fold", fold_idx, "*****")
        _reset_seeds()
        trained_model, evaluation = train_and_evaluate(
            os.path.join(ds_path, str(fold_idx)),
            model(),
            **get_default_hyperparams(),
            ckpt_path=os.path.join(ckpt_path, str(fold_idx)),
            tensorboard_path=os.path.join(tensorboard_path, str(fold_idx)),
        )
        trained_models.append(trained_model)
        trained_models_val_loss.append(evaluation)

    # Print final evaluation
    for fold_idx, evaluation in enumerate(trained_models_val_loss):
        print(f"Fold {fold_idx} evaluation:", trained_models_val_loss[fold_i
```

```
restore_ckpt = False, training from scratch
Epoch 1/3
3291/3291 [==============================] - ETA: 0s - loss: 62.8784
Epoch 1: saving model to ckpt/Baseline_cbr_mb/01-56.9121
3291/3291 [==============================] - 288s 81ms/step - loss: 62.8784
- val_loss: 56.9121 - lr: 0.0010
Epoch 2/3
3291/3291 [==============================] - ETA: 0s - loss: 57.7597
Epoch 2: saving model to ckpt/Baseline_cbr_mb/02-56.2310
3291/3291 [==============================] - 271s 82ms/step - loss: 57.7597
- val_loss: 56.2310 - lr: 0.0010
Epoch 3/3
3291/3291 [==============================] - ETA: 0s - loss: 55.4976
Epoch 3: saving model to ckpt/Baseline_cbr_mb/03-53.2088
3291/3291 [==============================] - 261s 79ms/step - loss: 55.4976
- val_loss: 53.2088 - lr: 0.0010
823/823 [==============================] - 21s 26ms/step - loss: 53.2088
Final evaluation: 53.208797454833984
```

It seems to be working, we obtain a loss of 53 % ! Still we have a quite high loss in the end, the best contestant for the challenge reached a loss around 20%.

A first idea to make it work better is to increase the number of epochs. Though for our project, we'll stay with algorithms that can be executed in a short time. Then let's just try to increase the number of epochs and to decrease the dataset, just to make sure that, in theory, increasing the number of epochs would be working. In other words, we'll try to reduce bias and increase the variance

Let's reduce the size of the training dataset :

In [ ]:
```
len(ds_train)
```

Out[ ]:
```
3291
```

In [ ]:
```
if not os.path.exists("data_cbr_mb_reduced"):
  os.makedirs("data_cbr_mb_reduced")

shutil.copytree("data_cbr_mb_cv/0/", "data_cbr_mb_reduced/0/")
```

Out[ ]:
```
'data_cbr_mb_reduced/0/'
```

In [ ]:
```
def get_default_hyperparams() -> Dict[str, Any]:
    """Returns the default hyperparameters for the training of the models. T
    - Adam optimizer with lr=0.001
    - MeanAbsolutePercentageError loss
    - No additional metrics
    - EarlyStopping and ReduceLROnPlateau callbacks
    - 100 epochs
    """
    return {
        "optimizer": tf.keras.optimizers.Adam(learning_rate=0.001),
        "loss": tf.keras.losses.MeanAbsolutePercentageError(),
        "metrics": [],
        "additional_callbacks": get_default_callbacks(),
        "epochs": 15,
    }
```

In [ ]:
```
ds_path = "data_cbr_mb_reduced"
model = Baseline_cbr_mb
cfv = False
n_folds = 5

# code for simple training/validation
if not cfv:
    _reset_seeds()
    trained_model, evaluation = train_and_evaluate(
        os.path.join(ds_path, "0"), model(), **get_default_hyperparams()
    )
    print("Final evaluation:", evaluation)

# code for cross-fold validation
else:
    trained_models = []
    trained_models_val_loss = []
    ckpt_path = f"ckpt/{model.name}_cv/"
    tensorboard_path = f"tensorboard/{model.name}_cv/"

    # Execute each fold
    for fold_idx in range(n_folds):
        print("***** Fold", fold_idx, "*****")
        _reset_seeds()
```

```python
        trained_model, evaluation = train_and_evaluate(
            os.path.join(ds_path, str(fold_idx)),
            model(),
            **get_default_hyperparams(),
            ckpt_path=os.path.join(ckpt_path, str(fold_idx)),
            tensorboard_path=os.path.join(tensorboard_path, str(fold_idx)),
        )
        trained_models.append(trained_model)
        trained_models_val_loss.append(evaluation)

    # Print final evaluation
    for fold_idx, evaluation in enumerate(trained_models_val_loss):
        print(f"Fold {fold_idx} evaluation:", trained_models_val_loss[fold_i
```

```
restore_ckpt = False, training from scratch
Epoch 1/15
3291/3291 [==============================] - ETA: 0s - loss: 62.9783
Epoch 1: saving model to ckpt/Baseline_cbr_mb/01-59.6077
3291/3291 [==============================] - 413s 118ms/step - loss: 62.9783
- val_loss: 59.6077 - lr: 0.0010
Epoch 2/15
3291/3291 [==============================] - ETA: 0s - loss: 57.5587
Epoch 2: saving model to ckpt/Baseline_cbr_mb/02-56.4090
3291/3291 [==============================] - 380s 115ms/step - loss: 57.5587
- val_loss: 56.4090 - lr: 0.0010
Epoch 3/15
3291/3291 [==============================] - ETA: 0s - loss: 55.8112
Epoch 3: saving model to ckpt/Baseline_cbr_mb/03-53.4596
3291/3291 [==============================] - 383s 116ms/step - loss: 55.8112
- val_loss: 53.4596 - lr: 0.0010
Epoch 4/15
3291/3291 [==============================] - ETA: 0s - loss: 54.6284
Epoch 4: saving model to ckpt/Baseline_cbr_mb/04-54.7580
3291/3291 [==============================] - 374s 114ms/step - loss: 54.6284
- val_loss: 54.7580 - lr: 0.0010
Epoch 5/15
3291/3291 [==============================] - ETA: 0s - loss: 54.3573
Epoch 5: saving model to ckpt/Baseline_cbr_mb/05-53.6658
3291/3291 [==============================] - 376s 114ms/step - loss: 54.3573
- val_loss: 53.6658 - lr: 0.0010
Epoch 6/15
3291/3291 [==============================] - ETA: 0s - loss: 54.2766
Epoch 6: saving model to ckpt/Baseline_cbr_mb/06-53.2839
3291/3291 [==============================] - 371s 113ms/step - loss: 54.2766
- val_loss: 53.2839 - lr: 0.0010
Epoch 7/15
3291/3291 [==============================] - ETA: 0s - loss: 54.3001
Epoch 7: saving model to ckpt/Baseline_cbr_mb/07-53.9859
3291/3291 [==============================] - 372s 113ms/step - loss: 54.3001
- val_loss: 53.9859 - lr: 0.0010
Epoch 8/15
3291/3291 [==============================] - ETA: 0s - loss: 54.2149
Epoch 8: saving model to ckpt/Baseline_cbr_mb/08-52.3203
3291/3291 [==============================] - 372s 113ms/step - loss: 54.2149
- val_loss: 52.3203 - lr: 0.0010
Epoch 9/15
3291/3291 [==============================] - ETA: 0s - loss: 53.0463
Epoch 9: saving model to ckpt/Baseline_cbr_mb/09-53.0660
3291/3291 [==============================] - 372s 113ms/step - loss: 53.0463
- val_loss: 53.0660 - lr: 0.0010
Epoch 10/15
3291/3291 [==============================] - ETA: 0s - loss: 50.5056
Epoch 10: saving model to ckpt/Baseline_cbr_mb/10-49.0217
3291/3291 [==============================] - 369s 112ms/step - loss: 50.5056
- val_loss: 49.0217 - lr: 0.0010
Epoch 11/15
3291/3291 [==============================] - ETA: 0s - loss: 48.8000
Epoch 11: saving model to ckpt/Baseline_cbr_mb/11-49.9452
3291/3291 [==============================] - 380s 116ms/step - loss: 48.8000
- val_loss: 49.9452 - lr: 0.0010
```

```
Epoch 12/15
3291/3291 [==============================] - ETA: 0s - loss: 51.4617
Epoch 12: saving model to ckpt/Baseline_cbr_mb/12-44.6818
3291/3291 [==============================] - 382s 116ms/step - loss: 51.4617
- val_loss: 44.6818 - lr: 0.0010
Epoch 13/15
3291/3291 [==============================] - ETA: 0s - loss: 46.8418
Epoch 13: saving model to ckpt/Baseline_cbr_mb/13-48.3854
3291/3291 [==============================] - 370s 112ms/step - loss: 46.8418
- val_loss: 48.3854 - lr: 0.0010
Epoch 14/15
3291/3291 [==============================] - ETA: 0s - loss: 48.2409
Epoch 14: saving model to ckpt/Baseline_cbr_mb/14-42.9410
3291/3291 [==============================] - 369s 112ms/step - loss: 48.2409
- val_loss: 42.9410 - lr: 0.0010
Epoch 15/15
3291/3291 [==============================] - ETA: 0s - loss: 47.3611
Epoch 15: saving model to ckpt/Baseline_cbr_mb/15-41.8691
3291/3291 [==============================] - 348s 106ms/step - loss: 47.3611
- val_loss: 41.8691 - lr: 0.0010
823/823 [==============================] - 30s 37ms/step - loss: 41.8691
Final evaluation: 41.869136810302734
```

We reach a result of 41.9% average loss.