To Pravin Varaiya and Jean, . . .

Studying a TimeSeries Regression Problem Through Automated Deep Neural Network Approaches

Jennie Lioris
ljenniemail@icloud.com,
jenniemail12@gmail.com

1 Introduction

A timeseries regression problem is considered aiming to forecast flight arrival delays from provided datasets involving information related to various flight features.

Hence, within this study the task of approximating a mapping function between the time ordered input information (considered data) to a continuous output variable (flight arrival delays) is performed.

Different Deep Neural Network (DNN) architectures are modeled. Through "adequate" automated tuning of specific hyperparameters associated with each model, the resulting topology (hypothesis space) within which possible solutions can be searched is defined. Among all possible topologies implied by a given hyperparametrized DNN model the optimal one(s), that is the most performant hyperparameter configuration among the examined ones, is (are) maintained.

Furthermore, a modular software appraising the performance of involved models is also developed. Thus, any DNN *tunable architecture*, available in the library, after been analyzed for different sets of hyperparameter combinations, will be evaluated through desired metrics and plots. Additionally, new models can be easily incorporated and experimented for desired datasets and possible study approaches.

The proposed study is organized as follows: § 2 discusses the conceived DNN models, § 3 presents the functions specifically developed for the proposed software allowing a model experimentation and appraisal, § 4 introduces directions for using the software, § 5 refers to the employed data treatment, necessary preliminary step before the information is at the availability of each model, § 6 discusses the baseline model while justifies the related utility of creating such a model, § 7 evaluates the introduced models for a specific study approach (involving a simultaneous exploitation of the entire dataset during a single study), § 8 appraises some of the suggested models according to a different study approach where a distinct study is proceeded for each "origin airport". Finally, § 9 resumes the suggested study while § 10 proposes new objectifs for further improvement of the suggested work.

2 Model Description

Efficient mapping from input to output data through a series of operations is aimed when handling timeseries regression problems.

In order to learn from a specific dataset, *assumptions* on this data are required. Different sets of considered hypotheses imply different acquired knowledge out of the same data.

A model topology determines the *assumptions* to be respected for the involved problem by structuring the space of possible data representations, the "hypothesis space" within which a solution is going to be searched.

Once the "hypothesis space" is defined, adequate "learning configuration" should be decided returning reliable outcomes.

With the purpose of a systematic and automatized exploration of the "hypothesis space", some simple model architectures are initiated. The principle respected all along this study, considers preferable to obtain automatized processes for *hyperparameter optimization*. Thus, refined hyperparameter values should be selected by the model itself for creating models able to adapt to new contexts and deal well with unseen events.

In what follows different model architectures are designed each one limiting the search within a constrained "hypothesis space". Furthermore, within each model, a first elementary effort is made, to automatize the model adjustment aiming at optimized results (good generalization level able to handle new contexts). Thus, different "topologies" are experimented though distinct model configurations allowing "learning" from various desired optimizers (feedback algorithms updating the network parameters-weights), objective functions to be optimized during the learning process and associated metrics to evaluate the resulting outcome. In addition, the system dimensioning will also be examined such as number of layers, units (whenever such a variable applies) as well other involved hyperparameters within layers, (e.g. the kernel size, activation function), optimizers and loss function etc.

Each conceived model is developed in file named "cc_fi_models_1".

However, one can create and try more elaborated models. Due to the modularity of the conceived tool, any new model can be easily integrated after being added in file "cc_fi_models_1". (also see § 4.1). This is one of the multiple goals of the proposed work. Creation of a tool able to optimize and appraise any desired model topology.

All models are following a graph-oriented architecture. They are created using the *Keras Functional API* ([1]) a modeling providing satisfactory flexibility. Thus, models with multiple inputs and/or outputs or involving non linear topologies can be considered.

In order to improve the model performance *KerasTuner* (see [2]) will be employed for selecting sets of optimized hyperparameter values (hyperparameter tuning). After examination of the space of possible decisions the best selected values will automatically be returned.

When designing a model various questions need to be answered regarding the involved hyperparameters such as:

- the number of layers to use

- the number of units or filters associated with each layer - depending upon the model architecture (dense, recurrent or convent)
- potential employment of a LayerNormalization layer
- use of (recurrent) Dropout/Dropout rate or other technique controlling overfitting
- selection of the model "optimiser" and the metric to monitor
- etc.

At a single run (or trial) the hyperparameter optimization process proceeds as follows:

1. selects a set of hyperparameter values among the desired ones
2. creates the associated model
3. learns the model parameters by fitting the new model to the train dataset
4. appraises the model performance using the validation dataset
5. define the next set of hyperparameter to evaluate

A specific algorithm selects the hyperparameter set after examining the connection between and performance on the validation dataset and different possible hyperparameter values.

Among the available built-in tuner algorithms, the *Bayesian Optimization* will be employed, [3]. Additionally, the *Gridsearch* algorithm of *KerasTuner* will also be tried ([4]).

The search space (space of feasible solutions) is going to be specified by subclassing the *HyperModel* class in order to achieve an adequate modular and reusable designs (see [5]).

The types of considered layer types in this study are:

- Dense, ([6])
- Convolutional 1D (Convents, Temporal Convolution), ([7])
- Recurrent (LSTM, GRU), ([8])
- Recurrent Bidirectional.

Layers can be associated with

- Dropout layers (regularization technique reducing overfitting [9])
- Recurrent Dropout (regularization technique reducing overfitting regarding the recurrent state of recurrent layers)
- LayerNormalization layers (for train acceleration and potentially reducing generalization error by controlling overfitting [10])

- MaxPooling1D layers (reducing the number of feature map coefficients while convolutional layers explore significantly large windows [11])
- GlobalAveragePooling1D layer (considering the average among time steps per feature dimension [12]).

As we are dealing with a regression problem the last layer will be a "Dense" one with a number of units equal to the number of time-steps (or future observations) to forecast.

Models comprised of three distinct layer categories are developed.

1. *Dense layered networks* (*or fully connected* as the units of each layer are connected to all other units). This type of network ignores existence of any specific data structure attempting to find relationships between any two input features.
2. *Convolution* 1*D layered networks* searching for local patterns (by applying the same transformation to different data patches). They are translation invariant meaning that a learned pattern can be recognized anywhere contrary to dense networks which they will have to *re*learn the same pattern whenever it appears.
3. *Recurrent Neural Networks* (RNN) comprised of LSTM or GRU layers. At each timestep a sequence of inputs is processed while the network state is considered (representing past information). Their advantage to *Convolution* 1*D layered* networks consists of detecting patterns that are not translation invariant. This could be the case of timeseries involving entries for which the reliability of an observations varies with the time (temporal translation variance).

Remark 2.0.1 *In this time-series problem modeling the number of entries used either for learning, validation or for inference does not necessarily corresponds to distinct time values as distinct flights can be delayed at the same time.*

2.1 Model Dense 1

This model creates a sequence of fully connected layers.

1. It starts by flattening the inputs
2. next, it tunes the number of dense layers. For each layer (except the last one) the following characteristics are tuned in the presented order,
   (a) the number of units is decided among a set of possible values varying between a min and a max value. As this study deals with a regression problem, the last "Dense Layer" will have as many units as the numbers of predictions (number of time steps for which the value of the desired variable should be forecasted)
   (b) the activation function is selected among set of desired activation functions. The last Dense layer will not be activated for preventing constraining the output. Leaving the last layer linear, the model learns to predict values of any range

3. thus, using the previously defined values and the Keras Functional API, a *keras model* is going to be created ([13]).

   For preparing the model training we need to define the following elements within the compilation step

   - an optimizer which will be selected among a list of desired ones (that is an algorithm specifying how the gradient of the loss function will update the involved parameters based on the training dataset, [14])
   - the learning rate of the optimizer is going to be determined by selecting a value between the range of the min and max predefined values (value controlling how much the weights will be updated according to the estimated error).
   - the loss function which will also be specified among a set of possible ones (function to be minimized along the training, [15])
   - metrics will be defined for evaluating the model performance on both training and validation sets, ([16]).

Remark 2.1.1

In this model for each Dense layer an activation function is selected. If a large number of trials is decided the tuner should automatically return whether each layer should have a distinct activation function or not. An alternative would be to create a new model where the activation function will be selected once and reused by all layers. The same holds true for all the following models.

2.2 Model Dense 2

This model is a variation of model defined in § 2.1. A data normalizing technique is introduced allowing smoother better generalization and faster training.

Hence, this model may involve a LayerNormalizaton after each Dense layer definition (whenever the tuner selects this option). Since sequence data are involved in the dealing problem, LayerNormalizaton is preferred to BatchNormalization ([17]). Within a batch each sequence is normalized independently from the others when LayerNormalizaton occurs contrary to a BatchNormalization which within a batch it normalizes each feature independently. Since LayerNormalization is independent of the batch size it suits to smaller sizes of batches. When a LayerNormalization is decided then the preceded "Dense" layer does not involve a bias vector (use_bias=False) since the output of this layer is going to be normalized.

2.3 Model Dense 3

Another variation of model defined in § 2.1 is considered. Instead of a Layer Normalization, the potential use of a Dropout layer (a regularization technique preventing or at least reducing overfitting) is here examined after the definition of intermediate Dense layers. If Dropout is decided then a pre-selected "dropout rate" (fraction of the features to take zero value) will be utilized (same value for all the Dropout layers allowing the model to properly propagate the learning error over time).

2.4 Model SimpleConv1D 1

This is a simple 1D convolutional model comprised of three "Conv1D" layers the first two of which are followed by "MaxPooling1D" layers (allowing input downsampling by taking the maximum value over a window of size equal to the value of parameter "pool_size").

A "GlobalAveragePooling1D" layer follows the last "Conv1D" layer for downsampling the average of the time per feature dimension. The layer series is completed by a linear (non activated) "Dense" layer involving as many units as the number of time-steps to predict.

Among the parameters of a "Conv1D" layer, parameters "filter" defining the dimension of the output space and and "kernel size" specifying the length of the convolution window are going to be defined.

For this model, each "Conv1D" layer is comprised of eight filters and associated with a "Relu" activation function.

The initial value of the kernel size equals 24 and progressively is divided by two for the following "Conv1D" layers (as sequences are downsampled by the use of "Maxpooling1D" layers).

The last layer is a linear "Dense" one with as many units as the number of time-step values to forecast.

The optimizer and loss function are automatically selected by the tuner which selects values among a list of desired ones. Finally, the learning rate will be tuned for values varying within a desired range of values.

2.5 Model SimpleConv1D 2

A more flexible Temporal Convolution model is now built comprised of the following ordered sequence of steps:

1. the number of layers, "filters", "kernel size" and "pool size" is (automatically) selected by the tuner

2. with respect to the defined number of layers, "Conv1D" layers will be constructed. Each "Conv1D" layer has a "kernel size" equal to the half of the previous one whenever this value is strictly positive, otherwise it keeps the previous value. In this model all "Conv1D" layers have the same kernel size and activation function "relu". However, the activation function should also be tuned in order to find the one adapted to the employed data

3. after each "Conv1D" layer, it is examined whether to apply a "LayerNormalization" layer

4. a "Maxpooling" layer is always created after 'Conv1D" (when a "LayerNormalization" layer is used the "Maxpooling" layer follows it (one can try to reverse the order by potentially adding a "LayerNormalization" layer after the "Maxpooling" layers). All "Maxpooling" layers have the same pool size selected at the beginning of the process

5. when the layer sequence is constructed a "Global Average Pooling " layer is employed (averaging time values per feature)

6. similarly to all the other models the layer sequence ends with a non-activated "Dense" layer having the number of time-steps to predict as number of units

7. finally, the values for optimizer, loss and the learning rate will be defined by the tuner (selecting ones within sets of desired possible alternatives to be examined).

Remark 2.5.1 (Potential Raised Error)

This approach may fail to run due to potential errors the creation of which mostly depends on the selected hyperparameter values (due to downsampling, pool size etc.). One has to define "improved" techniques when searching efficient values of the hyperparameters to be tuned.

In this work an elementary version of "Conv1D" layered model is presented.

For experimenting this version the range/set of the hyperparameter values should be carefully determined before proceed to any execution of the program. In case of errors the model can be rerun after initiating it with different values.

However, one of the intended objectifs of this work was to define efficient adaptive models. That is, creating a model able to automatically select compatible hyperparameter values respecting all imposed constraints. In case of impossibility one among the multiple multiple options could be that the model would itself select different values and/or modify the sets with the desired values for each hyperparameter if no adequate value could be found. Another alternative would be to create a model defining itself the feasible sets of values for the hyperparameters to be tuned and define a functional model (regarding both the dimensioning and choice of other operating algorithms) within each trial.

One can conceive various smart procedures for efficient decisions when defining a model. Due to multiple constraints this part of the study couldn't be completed on time and it is not included in this work.

2.6 Model RNN 1

This is a simple recurrent neural network (RNN) maintaining a state involving past information. It is comprised of a single LSTM layer of fourteen units (the number of utilized input features is twelve) and a linear "Dense" layer with as many units as the number of outputs (time-steps) to predict. The default activation function "tanh" is employed for the recurrent layer. The optimizer, loss function and the learning rate are tuned by the model selecting values belonging to predefined sets.

2.7 Model RNN 2

This model has an increased capacity as it creates two GRU layers (relatively simpler than LSTM layer), each layer of which is employing thirty two units and activated by the "tanh" function.

Since the network size is increased, to prevent the model from overfitting both *dropout* and a *recurrent dropout* are considered (with rate equal to 0.5 and 0.25 respectively) specifying the dropout rate for the input units (of the last "Dense" layer) and the recurrent dropout for the recurrent units respectively. In order for the model to properly propagate the learning error the same value of recurrent dropout rate is employed by each layer.

As all models using dropout layers for being regularized, this model too requires a longer training.

2.8 Model RNN 3

This model generalizes the one defined in § 2.7 in the sense that the number of recurrent layers (this model employs LSTM type of layers), the *dropout rate* and the *recurrent dropout rate* are also tuned (that is values automatically selected among predefined sets of desired values). For allowing the model to adequately propagate the learning error over time-steps the same values of (recurrent) dropout are going to be utilized whenever (recurrent) dropout is required. Additionally, the activation function is automatically selected and the same value is used for all the LSTM layers. However, one may examine whether the activation function should be decided within each layer creation. Furthermore, it is examined whether to employ a LayerNormalization layer (following each recurrent layer).

2.9 Model RNN 4

This model applies a bidirectional technique for exploiting the order sensitivity of the model developed in § 2.8. More precisely the same steps of model developed in § 2.8 are considered with the difference that bidirectional LSTM layers are employed instead. Each data sequence is processed according to the two possible orders (ascending and descending dates) before merging the outcomes (also see [18]). According to the context, additional patters may be detected which they were ignored when exploring a single time direction (increasing or decreasing dates).

2.10 Model RNN 5

This model is comprised of the same steps as the model developed in § 2.9 with the difference that now bidirectional GRU layers are experimented instead of LSTM ones.

3 Function Description

Aiming at emphasizing a modular architecture and reusability a set of functions is proposed each one defining a specific task. These functions are contained in file "cc_fi_fcts_1" and are going to be described.

3.1 fct_create_di_dataframes_per_origin

Function creating a dictionary of dataframes from an initial dataframe. Each dataframe corresponds to a specific value of variable "ORIGIN". The key of the dictionary is the name of the "ORIGIN" airport and the value is the dataframe with the corresponding entries to the "ORIGIN". Arguments of the function:

- val_dataframe: the dataframe containing all the information
- val_name_col_origin="ORIGIN", the name of the feature for which different dataframes will be created for each distinct value taken by this feature variable.

The default value of this variable is the feature "ORIGIN" as for the current study, distinct dataframes regarding the value of the origin airport are going to be considered

- val_name_col_1_sort="FL_DATE", the name of the first feature to sort each created dataframe
- val_name_col_2_sort="DEP_TIME", the name of the second feature to sort each created dataframe
- val_admissible_min_nb_observatios=80, the minimum number of entries required in order to create the associated dataframe with a given value of variable val_name_col_origin.

3.2 fct_vectorization_obj_cols_single_df

Function which vectorizes object type columns of a single dataframe. The employed encoder is LeaveOneOutEncoder [19] which is a target based encoder. The advantages of this encoder is that it doesn't increases the number of variables as it doesn't create additional variables (case of One Hot Encoder). However one should also try different encoding approaches.

Arguments of the function:

- val_dataframe: the dataframe
- val_li_name_cols_to_ignore: list with the features to be ignored during the vectorization
- val_li_name_cols_to_copy: the list with the features to be copied before the transformation so as to have their initial values after the encoding.

This function returns the encoded dataframe.

3.3 fct_measure_mean_observations_per_day_single_df

Function returning the number of mean observations per day.

Arguments of the function:

- val_name_df: the dataframe
- val_name_column_date: the name of the column with the dates (the considered dataframe corresponds to timeseries problem).

It returns the mean number of daily observations (flight departures).

3.4 fct_stand_single_df

Function standardizing the training samples of a single dataframe. In order to have small values with an comparable scale each time series will be standardized independently. The mean and standard deviation values of each feature of the training set will be computed. Then these values will subtracted and divided respectively, from each feature entry of the data.

Arguments of the function:

- val_name_df: the dataframe
- val_proportion_train_set: the proportion of the observations for the training set
- val_proportion_val_set: the proportion of the observations for the validation set

This function returns a list with the following elements:

- standardized data array
- number train samples: the number of training samples
- number val samples: the number of validation samples
- number test samples: the number of test samples
- mean: the mean value for the training set (per feature)
- std: the standard deviation value for the training set (per feature).

3.5 fct_examine_existence_null_vals_dataframe

Function examining the existence of missing (null) values for a list of dataframes.

It takes a single argument which is the list with the names of the dataframes to examine. It returns 0 or 1 if no dataframe or at least one dataframe has missing values respectively and also prints a message.

3.6 fct_plot_correlation

Function which plots the correlation heatmap using the Seaborn data visualization library.

Arguments of the function:

- v_data: the data to plot
- v_folder_figure: the folder to save the figure with the heatmap
- v_name_fig: the name of the file with the heatmap plot.

3.7 fct_creation_data_arrays_model_2

This function

- examines the existence of missing values in the provided dataframe
- creates the list with the target values (variable to inference)
- vectorizes *categorical* features
- plots the correlation heatmap
- computes the mean number of observations per day
- creates creates the standardized data (arrays).

Arguments of the function:

- var_dataframe: the dataframe
- var_folder_figure: the folder to save the figure
- var_proportion_train_set: the data proportion to allocate to the training set
- var_proportion_val_set: the data proportion to allocate to the validation set.
- The remaining arguments concern names of feature columns in the dataframe (ORIGIN, DEST, FL_DATE, DEP_TIME, ARR_DELAY, ["FL_DATE","ORIGIN"] as well the name of the figure to save the plot of the correlation heatmap). For the specific dataframes employed in this study one can use the default values.

This function returns a list with

- the standardized data
- the target variable
- the mean value of the daily observations.

3.8 fct_create_train_val_test_datasets_from_arrays

This function returns the train, validation and test sets in a form of sliding windows extracted from the timeseries data (returned value is an object of type tf.data.Dataset).

Each returned dataset is comprised of tuples (samples, targets). Each sample is a batch (set of observations) of a desired size comprised of a desired number of observations (timesteps) to use during the learning phase. Each target is also a batch comprised of the corresponding target values for each sample of a desired length.

The Keras function "keras.preprocessing.timeseries_dataset_from_array" ([20]) (alias of tf.keras.utils.timeseries_dataset_from_array) is employed for creating the tuples of datasets.

Arguments of the function:

- s_length_train: the sequence length indicating the time steps to consider for the training set
- s_length_target: the sequence length for the target variable corresponding to the number of time steps for the future forecasts
- s_stride: the stride of the sequence (period between successive output sequences)
- b_size: the batch size
- data: the data array to consider
- target: the target variable
- num_train_samples: the number of training samples
- num_val_samples: the number of validation samples
- shuffle_tr_s: whether to shuffle (or not) output samples of train sequence (True/False)
- shuffle_tr_t: whether to shuffle the targets of each train sequence (True/False)
- shuffle_v_s: whether to shuffle output samples of the validation sequence (True/False)
- shuffle_v_t: whether to shuffle the targets of each validation sequence (True/False)
- shuffle_t_s: whether to shuffle output samples of each test sequence (True/False)
- shuffle_t_t: whether to shuffle the targets of each test sequence (True/False).

3.9 fct_create_train_val_test_datasets_from_dataframe

This function creates train, validation and test sets of sliding windows from a dataframe.

It process the dataframe by calling function "fct_creation_data_arrays_model_2" defined in § 3.7. Once the standardized (and vectorized) arrays are available the train, validation and test sets are computed using function "fct_create_train_val_test_datasets_from_arrays" defined in § 3.8.

The arguments of the function are the arguments required by functions in § 3.7 and 3.8.

3.10 fct_get_best_epoch

Function which creates a *Keras model* for an Hyperparameters object [21]. It finds the best number of epochs to train this model. It returns the the best number of epochs for the training and the history object. This function will be employed to find the best number of epochs to retrain the best found models.

Arguments of the function:

- va_hp: the Hyperparameter object
- va_tuner: the tuner
- va_train_dataset: the train dataset (tuples (samples, targets))
- va_val_dataset: the validation dataset
- va_batch_size: the batch size
- va_metric_to_monitor_best_epoch_callbacks: the metric to monitor in the "callbacks" employed with "Earlystopping" for ceasing training when the metric stop, improving (see [22], [23])
- va_mode_callbacks: indicates the mode (min/ max etc.) according to which the training should stop (when the monitored metric stops decreasing/increasing etc.)
- va_patience_best_epoch_callbacks: indicates the number of epochs to continue the training without improvement. By the end of this period the training stops
- va_epochs=the (max) number of epochs to train the model.

3.11 fct_get_best_trained_model

Function which creates a model from a given set of hyperparameters, finds the best number of epochs for training the model using the function defined in § 3.10, retrains the model and returns the best trained model as well the history object resulting from the process of searching the best epoch and the best retrained model. As the model is retrained on a dataset comprised of the train and the validation sets(larger dataset) it is retrained for a "little" longer than the computed number of best epochs.

Arguments of the function:

- va_hp: the Hyperparameter object
- va_tuner: the tuner
- va_train_dataset: the train dataset (tuples (samples, targets))
- va_new_train: the new train dataset comprised of the initial training and validation set. One could define this set within the function definition. The purpose of having an explicit argument in this function is to remind the user that once the best number of epochs is found the model will be retrained in a larger dataset involving both train and validation datasets
- va_val_dataset: the validation dataset
- va_index_for_saving_best_model: an index for the recording of the best model when it is saved in memory. One can decide to have more than one best models in which case this index serves for distinguishing the distinct "best" models
- va_to_multiply_epoch_for_train_dur: the value to multiply the best number of epochs when retraining the model

- va_batch_size: the batch size
- va_metric_to_monitor_best_epoch_callbacks: the metric to monitor in the "callbacks" employed with "Earlystopping" for ceasing training when the metric stop, improving (see [22], [23])
- va_mode_callbacks: indicates the mode (min/ max etc.) according to which the training should stop (when the monitored metric stops decreasing/increasing etc.)
- va_patience_best_epoch_callbacks: indicates the number of epochs to continue the training without improvement, by the end of this number the training stops
- va_epochs=the max number of epochs for the training stage.

Remark 3.11.1 *A history object is a dictionary where the keys are the names of the loss functions and metrics and the values are the corresponding values of the loss functions and metrics per epoch. If both train and validation sets are employed then all values are indicated for each dataset separately.*

3.12 fct_search_best_model_using_tuner

This function finds:

- the best hyperparameters using a BayesianOptimization tuner (see [24])
- the best number of epochs to train the model(s) created with the best hyperparameters
- retrains the best model(s) for a little longer then the best number of epochs and returns the best retrained model(s).

A Tuner is a class in keras aiming at creating objects which search for the best hyperparameters sets. After creating a model for a given set of hyperparameter values, it trains this model and then evaluates it. This function employs one of the built-in Tuner subclasses in Keras, the "BayesianOptimization Tuner". According to bibliography this tuner is knows for proceeding to smart predictions. However one should also examine other types of tuners (see [25]).

Arguments of the function:

- val_di_tuners: the name of the dictionary with the defined tuners. In file "cc_fi_models_1" where the Keras models are defined a dictionary is also defined where the key is the name of tuner to use and the value is the tuner object
- val_key_tuner_class: the key of desired tuner to use (tuner define in the dictionary val_di_tuners)
- val_hypermodel: the Hypermodel object (see [5])
- val_objective_metric_for_tuner_to_optimize: the metric which the tuner will evaluates

- val_mode: the mode indicating the optimization direction of the objective function (min, max, auto), specified during the tuner definition
- val_max_trials: the max number of different models the tuner should try
- val_executions_per_trial: the number of executions per trial (in order to min the variance)
- val_directory: the name of the directory for recording the tuner logs
- val_metric_for_tuner_search_hp_callback: during the *tuner search* a callback object is defined involving an Earlystopping
- val_li_keys_tuners_optimizing_batch_size: dictionary the key of which indicates the key of the tuner which optimizes the batch size and the value is the related tuner object. This dictionary is defined in file "cc_fi_models_1". Whenever a tuner optimizing the batch size is created then its name must be included in this dictionary with an associated key number
- val_train_dataset: the train dataset (in form of tuples (samples, targets))
- val_val_dataset: the validation dataset (in form of tuples (samples, targets))
- val_test_dataset: the test dataset
- val_epochs_tuner_search: the number of epochs during the *tuner search*
- val_top_best_models: when the *tuner search* is completed a list with the best hyperparameter sets can be obtained. This variable indicates the desired number of "best" sets to be returned (number of elements in the returned list with the best hyperparameters)
- val_batch_size: the size of the batch (a fixed number is employed here however one should consider examining an alternative scheme involving batches of varying size adapted to the context)
- val_to_multiply_epoch_for_train_dur: the value to multiply the best number of epochs during the retraining of the best model on a larger dataset comprised of the previously considered train and validation sets
- val_metric_to_monitor_best_epoch_callbacks: the metric to monitor in the callbacks object defined during the searching of the best number of epochs to retrain each best model
- val_epochs_best_trained_model_search: the number of epochs to use when searching for the best number of epochs to train a best model
- val_overwrite: variable valuing True/False indicating whether the tuner will overwrite the data in the directory with the logs when starting a new search
- val_patience_during_tuner_search: the patience during the *tuner search* (number of admissible epochs without improvement) after which the model training stops

- val_verbose: variable tuning the verbosity during the tuner search
- val_mode_callbacks: the mode in the "callbacks" object used when searching the best trained model built using the best hyperparameters
- val_patience_best_epoch_callback: the patiente in the callback defined during the search of the best epoch for a best model

This function process as follows:

- creates a tuner for the given Hypermodel object
- searches the best hyperparameter values
- for each desired hyperparameter set the best retrained model is found. More precisely, after creating the model related to an optimized hyperparameter set the best number of epochs to train the best model is computed. The best model is retrained on the dataset comprised of both train and validation sets for a "little" longer than the best number of epochs.
- the best retrained model(s) is (are) returned (the number of retrained best models to be returned depends upon the user preferences).

3.13 fct_best_approaches_1

This function

- creates the train, validation and test datasets by calling function fct_create_train_val_test_datasets_from_dataframe defined in § 3.9 when a single dataframe is considered involving all values of all features. Thus, this function will be employed for the study involving all "ORIGIN" airports
- creates an Hypermodel object
- searches the best model(s) hyperparameter values by calling function fct_search_best_model_using_tuner defined in § 3.12.

3.14 fct_best_approaches_2

This function

- creates the train, validation and test datasets by calling function fct_create_train_val_test_datasets_from_dataframe defined in § 3.9 when a distinct dataframe for each value of a specific feature is considered. Each distinct "ORIGIN" is progressively examined when this function is called. The following steps are repeated for each dataframe
- it creates an Hypermodel object for a hyperparameter set
- it searches the best model(s) configuration(s) by calling function fct_search_best_model_using_tuner defined in § 3.12.

The arguments of this function are the arguments required by each one of the called functions.

3.15 fct_best_approaches

This function searches the best model(s) configuration. Function 3.13 or 3.14 is called depending upon the desired type of study, that is examining arrival delays considering only entries for each distinct "ORIGIN" or considering all entries for all features and values.

3.16 fct_create_di_destand_predictions

Thus functions computes the inferences using the test dataset for each best model. It returns the de-standardized values for each best model (by multiplying each inferences values by the standard deviation and adding te mean value).

Arguments of the function:

- val_di_best_models: the dictionary with the best models (key=id best retrained model, value=the best retrained model)
- val_test_dataset: the test dataset
- val_mean_value_train_dataset: the mean value of the variable to be forecasted
- val_std_train_dataset: the standard variation of the variable to be forecasted.

Remark 3.16.1 *In this model the target variable is also included in the training set*

3.17 fct_plot_predictions_test_set

Function which plots the forecasted values versus the true ones for a single model.

3.18 fct_plot_di_predictions_test_set

Function which plots the forecasted values versus the true ones for all the best models.

3.19 fct_plot_train_metrics_retrained_best_model

Functions which plots all the other metrics (loss, mean absolute error etc.) for the train and validation sets.

3.20 fct_arrange_metrics_test_set_per_model

This function creates a dictionary with the metrics using the results of the function fct_best_approaches defined in § 3.15. Te key of this dictionary is the id of the metric and the value is a list the ith element of which is the value of the metric for the ith model.

3.21 fct_plot_metrics_test_set_per_model

This function plots metrics for the test set.

3.22 fct_plot_graph_di_models

This function plots the model graph for each of the desired best models.

3.23 fct_analyze_results

This function analyzes the results of function fct_best_approaches defined in § 3.15 and plots the associated metrics and losses. One can redefine the function or at least extend it with any other desired plot.

4 Tool Employment

A simple version of a manual is going to be presented. Thus, the necessary steps to experiment a specific model are going to be described as well the required instructions for incorporating a new (Keras) model in the current software.

4.1 Model Extension

New models can be examined through the proposed tool (and this is one of the principle goals for the presented work). The employed programming language is Python 3 (version 3.10.5). Any new model should return an instance of Keras Model.

For adding a new Keras model class the next steps should be followed:

1. include the new model in file "cc_fi_models_1"
2. add the model name in the dictionary named "di_hypermodel" already defined in file 'cc_fi_models_1"
3. add the potential new arguments of the model to the rest of the models with the default value "None" (in order to be able to run any other model already included in the library of models). These arguments and their default values should also be included in the Hypermodel object named "hypermodel_j" constructed in file "cc_fi_fcts_1", function named *fct_best_approaches* (see 3.15)

4.2 Tuner Extension

As Tuners searching the batch size are also defined in this tool, every time a tuner is used its name has to be indicated in the dictionary named "di_tuners" defined in file "cc_fi_models_1". If also the tuner searches for the optimal batch size then the "key value" of the tuner has to be included in the list named "val_li_keys_tuners_optimizing_batch_size" defined in file "cc_fi_models_1".

Remark 4.2.1 *Perhaps this part of the tool has to be re-thinked. If a tuner is utilized in a model its the name has to be included in the dictionary "di_tuners" even for a tuner already defined in Keras while the Keras package is imported.*

4.3 Using the tool

In order to run the proposed software one needs to

1. fill the input variables in file “cc_fi_input_variables”
2. execute the file “c_fi_1”.

4.4 Input variables

In this section we will examine the necessary variables to fill in association with the desired model to appraise as defined in file “cc_fi_input_variables”.

4.4.1 Common variables for all presented models

The following variables should be initialized for any utilized model (file “cc_fi_input_variables”).

1. val_distinct_origins: variable indicating whether a single dataframe will be considered involving all entries for all values of all features (case when it values zero) or a single dataframe will be created considering all entries per each value of a given feature case when it values one)
2. val_admissible_min_nb_observatios: the min number of required observations for a dataframe to be considered (case when distinct dataframes will be created per each value of a give feature). This variable is mostly used for the approach considering a distinct dataframe per each “ORIGIN” value. There may be dataframes with too few entries (e.g. 20) in which case it may be preferable to have more data to use or particular approaches specifically destined for models with reduced data size. Consequently, as such models are not at present available in this work, it is preferable to ignore “very small dataframes”.
3. val_folder_figure: indicates the name of the folder with the figures (plots)
4. val_dataframe: the name of the dataframe to use. Here, it is considered that the provided dataframe is ready to use. Al necessary data treatment is already completed at a previous stage
5. val_min_nb_lay_model: the min number of layers to be selected. The network will have at least this number of layers
6. val_max_nb_lay_model: the max number of layers to be select. The network will have at most this number of layers.
7. val_step_nb_layers_model: the step to be considered when selecting the number of layers, a value between val_min_nb_lay_model and val_max_nb_lay_model
8. val_s_stride: value of the stride when creating train, validation and test sets using keras function *timeseries_dataset_from_array* it corresponds to the desired distance between consecutive sequences

9. val_batch_size: the size of the batch when creating train, validation and test sets

10. val_nb_past_seq_lengths: the number of sequence lengths sets to use during the learning phase. In the current model a sequence length is comprised of as many observations (timesteps) as the mean value of observations per month. If val_nb_past_seq_lengths> 1 the number of samples to use for learning is increased by a factor of val_nb_past_seq_lengths (sequence_length_learning=mean value of observations per month $\times$ val_nb_past_seq_lengths)

11. val_nb_future_seq_length: the number of sequence lengths sets to use for inference. In the current model a sequence length for forecast is comprised of the mean value of observations per month (e.g. for dataframe df_2009_1 this value equals to $1,106$ observations). When val_nb_future_seq_length> 1 the number of samples to use for inference is increased by a factor equal to val_nb_future_seq_length (sequence_length_forecast=mean value of observations per month $\times$ val_nb_future_seq_length)

12. val_s_length_train_model: variable indicating th desired sequence length to use for learning. When values "None" the mean number of observation per month it is going to be considered

13. val_s_length_target: variable indicating the desired sequence length to use for inference. When values "None" the mean number of observations per month will be employed

14. val_proportion_train_set: the proportion of the data to use for training

15. val_proportion_val_set: the proportion of data to use for validation. The remaining proportion of the dataset will be used for the test set

16. val_shuffle_tr_s,val_shuffle_tr_t,val_shuffle_v_s,val_shuffle_v_t,val_shuffle_t_s, val_shuffle_t_t: variables indicating whether to shuffle (True) the output samples created using keras function timeseries_dataset_from_array (train set, train taret, validation set, validation target, test set and test target or to respect the chronological order (False)

17. val_nb_last_output_classes_model: the number of units of the last layer which is of Dense type (regression problem)

18. val_li_optimizers_model: the list of the optimizers for the tuner

19. val_loss_fct_model: the list with the loss functions to consider

20. : val_metrics_model: the list with the metrics to evaluate the model performance

21. val_objective_metric_for_tuner_to_optimize: the metric for the tuner to optimize

22. val_max_trials: the max number of different model configurations to try during the tuner search

23. val_executions_per_trial: the number of times to train the same model in order to reduce the variance (one can average these results)

24. val_directory: path to a directory for storing the search results

25. val_overwrite: variable indicating whether to overwrite data in directory to start a new search

26. val_patience_during_tuner_search: variable indicating the number of epochs with no improvement after which training will be stopped in the callback during the tuner search

27. val_epochs_tuner_search: the number of epochs to train each considered configuration of a model (since an "Earlystoping callback" is defined a large number of epochs can be selected as training will stop when overfitting starts)

28. val_epochs_best_trained_model_search: the number of epochs when searching the best number of epochs for the best found models

29. val_verbose: variable expressing the verbosity level during the tuner search

30. val_top_best_models: the number of best models to consider after the tuner search

31. val_to_multiply_epoch_for_train_dur: parameter indicating the value to multiply the number of epochs when retrain the best model. Since larger data is now employed (train+validation) longer retrain will be applied

32. val_mode: the metric's optimization direction (min, max or auto) when defining the objective in the tuner instantiation

33. val_mode_callbacks: the direction to optimize the objective when defining a "callbacks" object during the search of the best epoch

34. val_metric_to_monitor_best_epoch_callbacks: the metric to monitor in the "callbacks" object during the search of the best epoch

35. val_metric_for_tuner_search_hp_callback: the metric to monitor in the "callbacks" object during the search of the best hyperparameters

36. val_patience_best_epoch_callbacks: the patience (number of epochs to tolerate without improvement) durning the search of the best number of epochs for the best model

37. val_li_label_1_for_plot_best_epoch:list titles for plots related to the best epoch

38. val_li_colors_for_plot_best_epoch: the list of colors to use for the plots

39. val_loc: the location for the plot legends

40. val_name_figure_metric_for_predicts: the name of the figure in the plot of the inferences versus true values

41. val_name_figure_find_best_epoch: the name of the figure with the metrics plots during the search of the best number of epoch s
42. val_name_figure_loss_best_epoch: the name of the figure with the plots during the search of the best epoch
43. val_title_best_epoch: the title of the plot during the search of the best epoch
44. val_pkl_filename_best_model: the name of the file where the best model is stored
45. val_pkl_filename_best_retrained_model: the name of the file where the best retrained model is stored
46. val_li_colors: the list with the colors names for the plots related to the best retrained model
47. val_li_markers: the list with the markers for the plots related to the best retrained model
48. val_name_figure_best_trained_model: the name of the figure with the plots of the best trained model
49. val_name_figure_loss_best_trained_model: the name of the figure with the loss plots for the best trained model
50. val_name_figure_plots_test_set: name of the figure with plots on the test set
51. val_name_file_plot_graph: the name of the figure with the graph of the best model
52. val_di_hypermodels: the name of the dictionary with the hypermodels. This dictionary is defined in file "cc_fi_models_1"
53. val_di_tuners: the name of dictionary with the tuners. This dictionary is defined in file "cc_fi_models_1"
54. val_key_hypermodel_class: the key of the hypermodel to be exploited (from dictionary val_di_hypermodels)
55. val_key_tuner_class: the key in the dictionary with the tuners specifying the tuner to use.

Remark 4.4.1

- *In variable val_dataframe a treated dataframe is considered with processed and cleaned data ready to be directly employed by a deep neural network architecture*
- *variables val_name_col_origin,val_li_name_col_to_copy,val_name_column_date, val_name_col_2_sort,val_name_target_variable,val_li_cols_to_ignore refer to features names that used during the data process (vectorization, standardization etc.) so the user should leave the default values as long as the dataframe has the features names as the one used here (dataframe for year 2009).*

- *val_mae_test_set, val_rmse_test_set: the values of the Mean and Routed Mean Absolute Error of the test set. They are set to None as for this version these values are not going to be used in the plots*
- *In § 4.4.1 the values of variables defined in 5, 6, 7 are required only by models tuning the number of layers of the involved network.*

4.4.2 Variables for the presented Dense layered models

The required variables for models comprised only on Dense layers are:

1. val_min_nb_units_model: the min number of units (for a single layer) to select
2. val_max_nb_units_model: the max number of units (for a single layer) to select
3. val_min_value_dropout_rate_model: the min admissible value for the dropout rate
4. val_max_value_dropout_rate_model: the max admissible value for the dropout rate
5. val_li_activ_fcts_model: the list with potential activation functions from which the automated model will select one. This variable is required only by the models which tune this variable and do not use a predefined value
6. val_min_val_learning_rate_optimizer: the min value of the learning rate
7. val_max_val_learning_rate_optimizer: the max value of the learning rate
8. val_step_nb_units_model: the step when selecting the number of units
9. val_step_dropout_rate_model: the step when selecting the dropout rate. This variable requires initialization only for the models tuning the dropout rate
10. val_step_recurrent_dropout_rate_model: the step when selecting the value for the recurrent dropout rate. This variable requires initialization only for the models tuning the recurrent dropout rate.

Among these variables only the ones required by each dense model need to be initialized. Thus, for example if a model does not tune the learning rate then no value is required for variables val_min_val_learning_rate_optimizer, val_max_val_learning_rate_optimizer.

4.4.3 Variables for the defined Conv1D layered models

The required variables for models comprised only on Conv1D layers are:

1. val_min_nb_filters_conv1d: the min admissible value for the filters
2. val_max_nb_filters_conv1d: the max admissible value for the filters
3. val_min_nb_kernel_size_conv1d: the min admissible value for the kernel size

4. val_max_nb_kernel_size_conv1d: the max admissible value for the kernel size
5. val_min_pool_size: the min value for the pool size for the Maxpooling layer
6. val_max_pool_size: the max value for the pool size for the Maxpooling layer
7. val_min_val_learning_rate_optimizer: the min value of the learning rate
8. val_max_val_learning_rate_optimizer: the max value of the learning rate
9. val_step_pool_size: the step size when selecting the value of the pool size
10. val_step_nb_kernel_size_conv1d: the step when selecting a value for the kernel size.

Similarly to the case of Dense layered models among these variables only the ones employed by the desired Cond1D model need to be provided.

Remark 4.4.2 *Model "SimpleConv1D_model_1" tunes only the loss function, the optimizer and the related learning rate. All the remaining variables (regarding, number of layers, filters and kernel size) are fixed predefined valued variables for this model.*

4.4.4 Variables for RNN layered models

The required variables for models comprised only on RNN layers are:

1. val_min_nb_units_model: the min number of units (for a single layer) to select
2. val_max_nb_units_model: the max number of units (for a single layer) to select
3. val_min_value_dropout_rate_model: the min admissible value for the dropout rate
4. val_max_value_dropout_rate_model: the max admissible value for the dropout rate
5. val_min_val_learning_rate_optimizer: the min value of the learning rate
6. val_max_val_learning_rate_optimizer: the max value of the learning rate
7. val_step_nb_layers_model: the step when selecting the number of layers
8. val_step_nb_units_model: the step when selecting the number of units
9. val_step_dropout_rate_model: the step when selecting the dropout rate
10. val_step_recurrent_dropout_rate_model: the step when selecting the value.

As with all the other models values for variables utilized by a given RNN model are necessary. Moreover, some RNN models are defined with fixed values for some of these variables (e.g.model defined in § 2.6 etc.).

5 Data Preparation

5.1 Data Description

The preliminary stage regarding the data treatment is going to be discussed. The related work is included in file "fi_creat_df".

The employed dataset is the one corresponding to year 2009 file named "2009.csv" and located in the folder "data_new_1". Due to the increased size of data and computer limitations a single year is going to be studied. However, the proposed methodology remains valid for any other year or when involving more than one years. This latter possibility would be very profitable to be explored but requires a powerful computer configuration.

One could read the data directly from the website (see [26]). Nevertheless, in this work the data is locally stored (ten yearly data-files, from year 2009 to year 2018 included with a total size 7.62 gigabytes). The reason for this decision is to prevail the risk of potential disruptions to internet access (internet providers updates, social events etc.) as well possible data withdrawal from internet etc. However, in case of data updates one should be aware of the data version employed but this hold trues whether data are read from local storage or internet availability.

The provided features are:

1. FL_DATE: date of the flight (yy/mm/dd)
2. OP_CARRIER: airline identification name
3. OP_CARRIER_FL_NUM: the flight number
4. ORIGIN: the code name of the departure location of the flight
5. DEST: the code name of the arrival location of the flight
6. CRS_DEP_TIME: expected departure time
7. DEP_TIME: actual departure time
8. DEP_DELAY: total delay on departure (in minutes)
9. TAXI_OUT: the time duration elapsed between departure from the origin airport gate and wheels off
10. WHEELS_OFF: the time at which the wheels of the aircraft leave the ground
11. WHEELS_ON : the time at which the wheels of the aircraft touch the ground
12. TAXI_IN: the elapsed time duration between wheels-on and gate arrival at the destination airport
13. CRS_ARR_TIME: expected arrival time
14. ARR_TIME: actual arrival time
15. ARR_DELAY: total delay on arrival (in minutes)

16. CANCELLED: cancelled flight (1 for cancelled flights)
17. CANCELLATION_CODE: reason for flight cancellation:
    - A - Airline/Carrier
    - B - Weather
    - C - National Air System
    - D - Security
18. DIVERTED: aircraft landed on a non scheduled airport
19. CRS_ELAPSED_TIME: expected time duration of the flight trip
20. ACTUAL_ELAPSED_TIME: the sum of the three features AIR_TIME, TAXI_IN and TAXI_OUT
21. AIR_TIME: the duration between the values of wheels_off and wheels_on
22. DISTANCE: the distance between the origin and destination airports
23. CARRIER_DELAY: delay caused by the airline (in minutes)
24. WEATHER_DELAY: delay caused by the weather conditions
25. NAS_DELAY: delay caused by air system
26. SECURITY_DELAY: delay caused by security
27. LATE_AIRCRAFT_DELAY: delay caused from late arrival of the aircraft assigned to operate that particular flight, see [27]

The number of total entries for the year 2009 is $6,429,338$.

Among all the provided operating carriers United Airlines (UA) will be only considered due to technical limitations (limited computer memory available). Consequently, the number of entries is reduced to $376,272$. However, one should consider more companies especially Southwest Airlines (WN) and American Airlines (AA) as they have an important number of flights and may interact to each other.

5.2 Data Treatment

The total number of missing values is $45,362$. Table 1 illustrates the number of missing values per feature where the total number of missing values is $45,362$.

Since cancelled flights will be delayed anyhow (regarding the initial expected arrival time) these flights will not be considered in the study. Additionally, diverted flights will also be delayed so the related entries are also removed from the considered dataset. Examination of the remaining dataset shows that there is no missing value for any feature.

The necessary procedures for the data treatment are available in file “fi_creat_df”.

A first visualisation of the numerical data is presented in Figure 1 representing. the evolution of each numerical feature over time.

Table 1: Missing Values - Before Data Treatment

Variable (Feature)	Number Missing Values
FL_DATE	0
OP_CARRIER	0
OP_CARRIER_FL_NUM	0
ORIGIN	0
DEST	0
DEP_TIME	5966
DEP_DELAY	5966
TAXI_OUT	6177
TAXI_IN	6301
ARR_DELAY	6984
CANCELLED	0
DIVERTED	0
CRS_ELAPSED_TIME	0
ACTUAL_ELAPSED_TIME	6984
AIR_TIME	6984
DISTANCE	0

It is of interest to examine the data (linear) correlation which is illustrated in Figure 2. One observes that some features are relatively highly correlated. Nevertheless, in this study correlated features will still be considered. Due to the modularity of the provided tool, one can easily proceed to an alternative study where only uncorrelated variables will be involved.

In Figure 3 the pairwise relationship between any two features is shown. One can see the linear correlation between some features in accord with Figure 2.

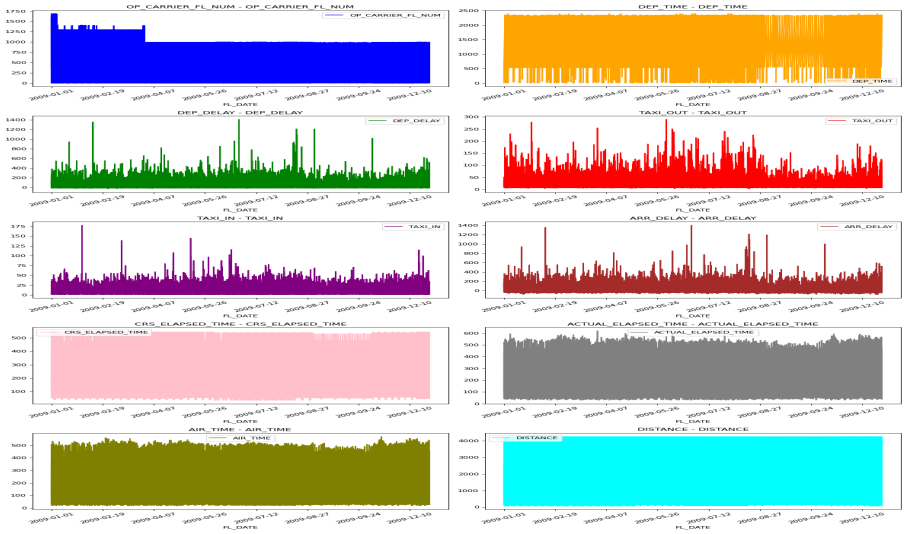

Figure 1: Numerical Features Visualisation.

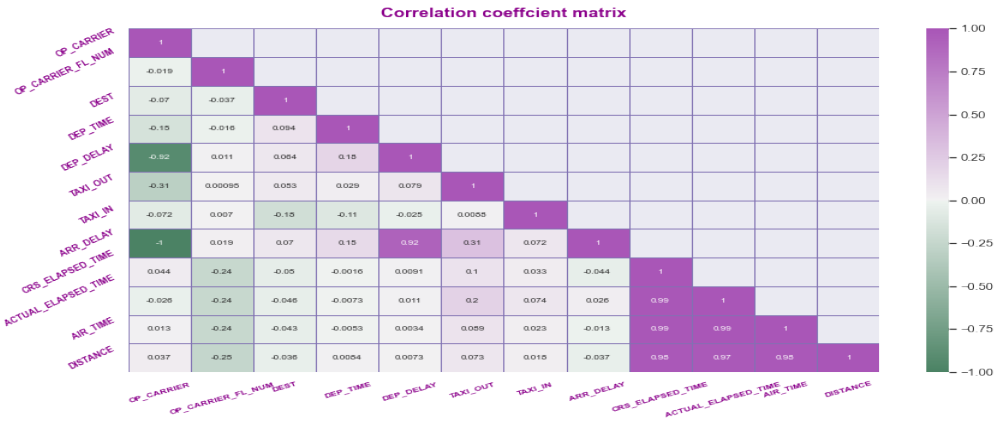

Figure 2: Data Correlation Heatmap.

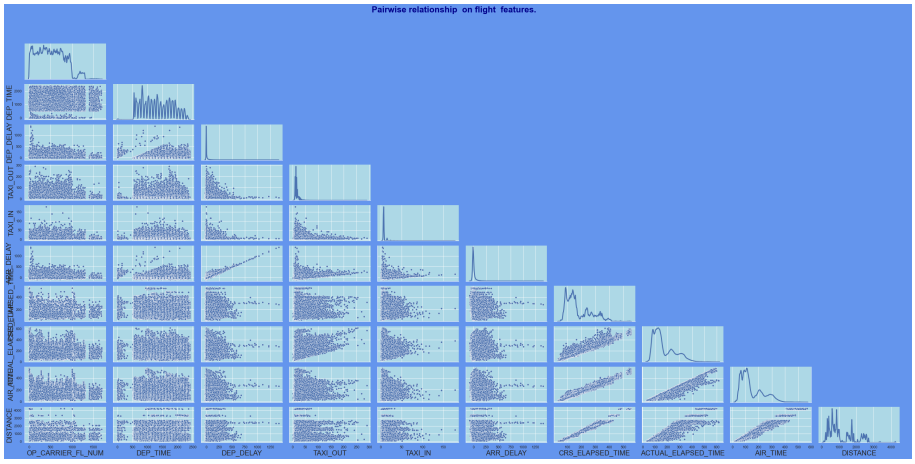

Figure 3: Pairwise Data Relationship.

Remark 5.2.1

- *An important step of the data preparation before using it to any model is the vectorization of the object typed variables as well the data standardization. This part of the process occurs when the training, validation and test sets are defined (see functions:fct_vectorization_obj_cols_single_df, fct_stand_single_df,fct_creation_data_arrays_model_2, fct_create_train_val_test_datasets_from_dataframe defined in § 3).*

- *Feature engineering is another useful data process consisting of using personal knowledge and inspiration for data transformation. This is an interesting part which may lead to significant result improvements. For example, one could apply feature engineering to process the values of the flying date etc. However, this technique is not used in the presented work. At present flying dates are only sorted chronologically. As mentioned before, due to the modularity of the tool, it is simple to proceed to a feature engineering before providing the related dataframe to a model.*

- *After data treatment all the remaining entries are going to be taken into consideration including outliers. As outliers may represent novelty even if they are associated with a weak probability they will be involved in the study in order to allow each model to learn the varying behavior of the features. However, a study excluding outliers can be realized. Due to the modularity of the proposed software, the dataframe to be considered is just an input variable for the tool, the name of which depends on the user. Information of any desired dataframe can be studied.*

6 Baseline Model

6.1 Introducing the Baseline Model and the Involved Performance

Aiming at examining whether it is reasonable to employ advanced models involving approaches of artificial intelligence, an intuitive model deploying the basic level of practical knowledge is going to be considered. The utility of each proposed model will be appraised by comparing the implied evaluation metrics with the ones of the baseline model. In other words, the baseline model will serve as reference solution as no other answer to the considered problem is available.

A simple and approximatively reasonable way to estimate the flight arrival delays is to compute the average flight arrival delay per origin airport (departing location of the flight). Knowing an average delay value for a given airport one can form an idea about an expected value for a flight delay.

Inferences will be made on the validation and test sets. For each origin airport and for both validation and test sets, the *mean absolute error* is going to be considered measuring the absolute value of the difference between true and estimated values of the flight arrival delays.

Finally, the mean value of the *mean absolute error* over all origins will be computed for the validation ans test sets.

For simplicity and in order to be in accordance with all the future experimentations of the new proposed models, data from year 2009 are employed. The proportions of the train, validation and test sets are 0.5, 0.25, 0.25 respectively. Thus, among the total available $369,288$ entries, $184,644$ are allocated to the training set while each of the validation and test sets receives $92,322$ observations. Each entry (observation) is comprised of available information regarding fourteen distinct flight features.

Hence, after computations, the *mean absolute error* of the considered baseline model for the validation set equals to 22.3 while this value for the test set is 23.

Remark 6.1.1

- *Within this study the considered dataset is divided into three subsets: the train, validation and test sets. The proportion of each subset is defined by the user. Since a timeseries problem is handled, each subset respects the time ordered entry sequences.*
- *Before utilized the considered dataset of year* 2009 *is processed. More analytical explanation about data treatment are provided in § 5 as all proposed models employe the same data information.*
- *Different baseline models can be conceived especially when involving data for more than one years. In that case one could employ average values of the same month per year. Alternatively, another option could be to consider the average delay of the last* n_d *days where an appropriate value of number* n_d *has to be determined.*

6.2 Baseline Model, Technical Report

The related code developing the proposed Baseline model is available in file "fi_BASELINE_MODEL_MEAN_PER_ORIGIN".

The mean value of the "mean absolute error" for all origin airports is computed for the validation and test sets using function defined in § 6.2.1.

6.2.1 fct_predict_ar_delay_baseline_model

This function charged for computing the average delay per origin airport.

It groups the considered dataframe per distinct origin airport (by creating sub-dataframes involving only flights departing the considered airport). Then, the mean value of feature "ARR_DELAY" is computed.

It receives the following arguments:

1. var_train_df: the dataframe involving observations of the training set
2. var_df_for_predictions: the dataframe to be employed for the inferences
3. var_name_group_feature="ORIGIN" : variable indicating the name of feature according to which the sub-dataframes will be created. The default value is feature "ORIGIN".

4. var_name_feature_to_predict="ARR_DELAY": the name of feature to be considered for inferences. The default value is the arrival delay ("ARR_DELAY").

This function returns the mean value over all origins of the "mean absolute error" computed for each origin airport.

7 Model Experimentation- Case Learning from all Origins

A complex spatiotemporal problem aiming at studying flight arrival delays is considered. In this approach, data of a single airline is employed, United Airlines (UA), involving flights departing and arriving from/to various airports over time, covering year 2009. The variable "arrival delay" takes real values, positive when the flight arrives later than the expected time and negative when the flight arrives earlier then the planned arrival time.

All considered experiments are operated through an iMac 3.8GHz, employing Ventura 13.4.1 macOS. The software is developed in "Python 3" programming language. The utilized versions of "Jupyter Notebook" packages are:

- IPython : 8.4.0
- ipykernel : 6.15.1
- ipywidgets : 7.7.1
- jupyter_client : 8.2.0
- jupyter_core : 5.2.0
- jupyter_server : 2.6.0
- jupyterlab: 4.0.2
- nbclient: 0.6.6
- nbconvert: 6.5.0
- nbformat: 5.4.0
- notebook: 6.4.12
- qtconsole: 5.3.1
- traitlets :5.9.0

Problem Statement

In this study, the assumption of *non independence* between the distinct values of feature "ORIGIN" is examined. More precisely, the learning phase of any model configuration will employ all possible values of all features in order to inference the forthcoming arrival delays. The estimated values will still be real numbers in order to forecast positive and negative arrival delays. Consequently, each proposed model will be experimented while fed by all available processed data. Using information on the

past n_p entries, the arrival delays for the future n_f flights are aimed to be forecast. The values of n_p and n_f are selected by the user. For the current experimentations $n_p = n_f =$ mean number of observations (flight departures from a given origin) per day. However, one may decide of different values corresponding to greater or smaller values for n_p and/or n_f.

Due to restrained computer ressources and in order to obtain the results within a very short time a small number of trials is selected for each possible model configuration. Thus, a limited number of hyperparameter combinations is experimented. For the same reason, small values are selected for other input variables (necessary to run the tool). A short training duration is allowed despite the definition of an *Early stopping* process interrupting the learning stage when generalization starts decreasing.

Consequently, the presented values of the loss function and performance metrics should not appraise the model potential especially when longer training durations are required for convergence (even more when Dropout layers are involved which are associated with slower convergence). Furthermore, additional values should be tried for the input variables (initialized in file "cc_fi_input_variables", also see § 4). Hence, smaller/larger values for the number of layers, units, the sequence length expressing the number of timesteps to use for learning or inferencing should be tried (e.g. should the model learn using past monthly, yearly, weekly, daily, hourly information ? see also § 10).

The explanation for all variables required to be initialized before any execution is available in § 4.

Remark 7.0.1

1. *In the presented experiments predefined layer activation functions included in Keras API are employed. However, one can try more advanced ones such as "LeakyReLU" or employ customized activation functions especially when the network state is desired to be taken into consideration, see [28].*

2. *Similarly to the case of the layer activation functions, predefined optimizers are employed in all model experimentations, available in Keras API, see [29].*

7.1 Common Variable Values For All Models

The following input variables have the same values for all the presented experiments, (initialized in file "cc_fi_input_variables", also see § 4).

1. val_batch_size$= 256$
2. val_nb_past_seq_lengths$= 1$
3. val_nb_future_seq_lengths$= 1$
4. val_proportion_train_set$= 0.5$
5. val_proportion_val_set$= 0.25$.

   The remaining proportion 0.25 is allocated for the test set
6. val_min_val_learning_rate_optimizer$= 1e-4$

7. val_max_val_learning_rate_optimizer= $1e-2$
8. val_loss_fct_model= [“mse'']
9. val_metrics_model= [“mae'']
10. val_objective_metric_for_tuner_to_optimize= “val_mae''
11. val_mode= “min''
12. val_mode_callbacks= “min''
13. val_top_best_models= 2
14. val_metric_to_monitor_best_epoch_callbacks= “val_loss''
15. val_patience_best_epoch_callbacks= 10
16. val_di_tuners=di_tuners
17. val_executions_per_trial= 2
18. val_overwrite= $True$
19. val_patience_during_tuner_search= 5
20. val_directory=“flight_del_kt_test"
21. val_name_figure_metric_for_predicts=“fig_inferences”
22. val_mae_test_set=None
23. val_rmse_test_set=None
24. val_name_figure_find_best_epoch=“fig_plot_metrics_search_best_epoch_best_retrained_model"
25. “fig_plot_metrics_search_best_epoch_best_retrained_model”
26. val_name_figure_loss_best_epoch=“Loss_Model”
27. val_verbose= 2
28. val_di_hypermodels=di_hypermodels
29. val_li_keys_tuners_optimizing_batch_size=fct_li_keys_tuners_tuning_batch_size()

Remark 7.1.1

1. *In the proposed experiments a single value for loss function is employed, the “mean squared error”, available in Keras API, as a very small number of trials is imposed. In previous runs the following list of loss functions was employed* [“$mean\_absolute\_percentage\_error''$, “$mean\_squared\_logarithmic\_error''$, “$mae''$]. *However, one may try more and/or different losses, even employ customized ones, see [30], allowing the model make different selections within the trials.*

2. *In the considered experiments a single function is considered as metric, the "mean absolute error". Similarly to the case of the loss functions more or different evaluation metric functions can be tried even customized ones, see [31]. In previous executions the following values have been tried the results of which are not included in this work due to time constraints.*

$$[tensorflow.keras.losses.Huber(name = \text{“}huber\_loss\text{”}),$$
$$\text{“}mean\_squared\_logarithmic\_error\text{”},$$
$$\text{“}mean\_absolute\_percentage\_error\text{”},$$
$$tensorflow.keras.metrics.RootMeanSquaredError(name =' rmse'),$$
$$tensorflow.keras.metrics.MeanSquaredLogarithmicError($$
$$name = \text{”}mean\_squared\_logarithmic\_error\text{”}].$$

3. *Each suggested model requires a tuning of the optimizer learning rate. However, when "adam" is selected as optimizer this operations is not really required as "adam" adapts the learning rate.*

7.2 Experimenting Model Dense 1-case BayesianOptimization Tuner

In this section the Dense model described in § 2.1 is going to be considered. As all the "Fully Connected" architectures the data is flattened before given to the network consequently the time order is ignored. Thus, the information included within each batch is transformed into a vector and treated as a single input.

Some of the input values when running this model are presented. The related file "cc_fi_input_variable_m1_k1" is included in folder named "FI_MODEL_RESULTS".

1. val_min_nb_lay_model= 2
2. val_max_nb_lay_model= 8
3. val_min_nb_units_model= 14
4. val_max_nb_units_model= 17
5. val_li_activ_fcts_model= $[' relu',' elu',' selu']$
6. val_li_optimizers_model= [“$rmsprop$”, “$adam$”, “$adagrad$”]
7. val_max_trials= 4
8. val_epochs_tuner_search= 24
9. val_epochs_best_trained_model_search= 20
10. val_key_hypermodel_class= 1
11. val_key_tuner_class= 1 (BayesianOptimization tuner).

Hereafter, the different considered model configurations are shown. Since four max trials are required, one observes four model architectures, one per trial. Hence, for the first trial (named trial 0) the "summary" describing the model hyperparameter values is as follows:

1. seven layers are decided
2. the first layer (layer 0) is comprised of 15 units and is activated using the "selu" activation function
3. the second layer (layer 1) is comprised of 16 units and is activated using the "relu" activation function
4. the third layer (layer 2) is comprised of 14 units and is activated using the "relu" activation function
5. the fourth layer (layer 3) is comprised of 14 units and is activated using the "relu" activation function
6. the fifth layer (layer 4) is comprised of 14 units and is activated using the "relu" activation function
7. the sixth layer (layer 5) is comprised of 15 units and is activated using the "elu" activation function
8. the seventh layer (layer 6) is comprised of 15 units and is activated using the "elu" activation function
9. the score of this model values $25,133924$.

Similarly for the other created models corresponding to trials 1, 2, 3.

```
val_metric_for_tuner_search_hp_callback,
v_li_keys_tuners_optimizing_batch_size=\
val_li_keys_tuners_optimizing_batch_size,
v_epochs_tuner_search=val_epochs_tuner_search,
v_top_best_models=val_top_best_models,
v_batch_size=val_batch_size,
v_to_multiply_epoch_for_train_dur=\
val_to_multiply_epoch_for_train_dur,
v_metric_to_monitor_best_epoch_callbacks=\
val_metric_to_monitor_best_epoch_callbacks,
v_epochs_best_trained_model_search=\
val_epochs_best_trained_model_search,
v_overwrite=val_overwrite,
v_patience_during_tuner_search=\
val_patience_during_tuner_search,
v_verbose=val_verbose,
v_mode_callbacks=val_mode_callbacks,
v_patience_best_epoch_callbacks=\
val_patience_best_epoch_callbacks,
v_pkl_filename_best_model=\
val_pkl_filename_best_model,
v_pkl_filename_best_retrained_model=\
val_pkl_filename_best_retrained_model
)
```

```
Trial 4 Complete [00h 10m 58s]
val_mae: 22.547080993652344

Best val_mae So Far: 20.815689086914062
Total elapsed time: 01h 07m 04s
INFO:tensorflow:Oracle triggered exit

Results summary
Results in flight_del_kt_test/untitled_project
Showing 10 best trials
Objective(name="val_mae", direction="min")

Trial 2 summary
Hyperparameters:
num_layers: 8
units_lay_0: 17
```



```
activation_0: selu
units_lay_1: 14
activation_1: relu
optimizer: adagrad
loss: mse
lr: 0.00020291889451666428
units_lay_2: 14
activation_2: relu
units_lay_3: 14
activation_3: elu
units_lay_4: 14
activation_4: elu
units_lay_5: 17
activation_5: selu
units_lay_6: 17
activation_6: relu
units_lay_7: 14
activation_7: relu
Score: 20.815689086914062

Trial 0 summary
Hyperparameters:
num_layers: 7
units_lay_0: 15
activation_0: selu
units_lay_1: 16
activation_1: relu
optimizer: adagrad
loss: mse
lr: 0.0044558929106897766
units_lay_2: 14
activation_2: relu
units_lay_3: 14
activation_3: relu
units_lay_4: 14
activation_4: relu
units_lay_5: 14
activation_5: relu
units_lay_6: 14
activation_6: relu
Score: 22.395140647888184
```



```
Trial 3 summary
Hyperparameters:
num_layers: 4
units_lay_0: 14
activation_0: selu
units_lay_1: 17
activation_1: selu
optimizer: adam
loss: mse
lr: 0.00012386292426143545
units_lay_2: 15
activation_2: selu
units_lay_3: 16
activation_3: relu
units_lay_4: 15
activation_4: selu
units_lay_5: 16
activation_5: elu
units_lay_6: 17
activation_6: elu
units_lay_7: 15
activation_7: relu
Score: 22.547080993652344

Trial 1 summary
Hyperparameters:
num_layers: 7
units_lay_0: 16
activation_0: relu
units_lay_1: 16
activation_1: selu
optimizer: rmsprop
loss: mse
lr: 0.0012347565315366372
units_lay_2: 17
activation_2: relu
units_lay_3: 15
activation_3: elu
units_lay_4: 15
activation_4: elu
```



```
units_lay_5: 15
activation_5: elu
units_lay_6: 15
activation_6: elu
Score: 25.13392448425293
Results Summary of the tuner None

IN FCT fct_search_best_model_using_tuner,    END TUN
ER SEARCH FOR FINDING BEST MODEL

In FCT fct_search_best_model_using_tuner,    val_top
_best_models 2


IN FCT fct_search_best_model_using_tuner,        hyp
erparameter number: 1
IN FCT fct_search_best_model_using_tunerr,       we
will search for the best trained model THAT IS THE B
EST EPOCH        AND RETRAIN THE BEST  MODEL using f
unction         get_best_trained_model
WE START FCT BEST_TRAINED_MODEL BY SEARCHING BEST EP
OCH
 IN FCT BEST_EPOCH, WE START SEARCH BEST EPOCH FOR H
P
In fct best_epoch, va_metric_to_monitor_best_epoch_c
allbacks val_loss

Epoch 1/20
713/713 [==============================] - 32s 44ms/
step - loss: 1558.6650 - mae: 21.8770 - val_loss: 14
42.5912 - val_mae: 20.8175
Epoch 2/20
713/713 [==============================] - 30s 43ms/
step - loss: 1558.5944 - mae: 21.8792 - val_loss: 14
42.5472 - val_mae: 20.8208
Epoch 3/20
713/713 [==============================] - 31s 43ms/
step - loss: 1558.4688 - mae: 21.8822 - val_loss: 14
42.4558 - val_mae: 20.8263
Epoch 4/20
713/713 [==============================] - 31s 43ms/
```

According to the value of variable val_top_best_models, two best models are considered.

Figures 4, 5 provide the summary of each created model. Observe that the output of the last (Dense) layer is comprised of $1,106$ elements. That is justified by the fact that estimation of the arrival delay for the $1,106$ future time steps is required (see file "cc_fi_input_variables", § 4, variable val_nb_future_seq_length).

Figures 6, 7 illustrate the evolution of the Mean Absolute Error (MAE) over epochs related to the training set when retraining the best models 1 and 2 respectively for a little longer than the best number of epochs.

Figures 8, 9 illustrate the evolution of the Loss for the training set when retraining the best models 1 and 2 respectively. One sees how the value of the loss function decreases as the training progresses (that is, as the number of epochs increases).

Figures 10, 11 depict how the MAE evolves with the number of epochs for both train and validation sets (blue, red curve respectively) when searching the best number of epochs to retrain the best models 1 and 2. As Figure 10 shows, regarding the best model 1, the MAE of the training and validation set present a similar behavior. A similar observation holds true for the best model 2.

Figures 12, 13 show the evolution of the loss for the train and validation sets when searching the best number of epochs to retrain the best models 1, 2 respectively. A faster decrease of the train loss regarding the validation loss is observed for both models. Notice how the validation loss (pink curve) for the best model 2 practically ceases to decrease after a few epochs (Figure 13).

Figures 14, 15 present the Loss and MAE on the test set respectively for each one of the two best models.

The best model 2 presents smaller values for both the loss and MAE metrics.

Thus, regarding the test set:

- the best model 1 is associated with a loss equal to 1132.73 and MAE equal to 21.29
- the best model 2 is associated with a loss equal to 1092.264 and MAE equal to 20.172.

These are the values depicted in Figure 14.

Figure 16 illustrates $1,106$ forecasted arrival delays on the test set (yellow curve) versus the true values (future arrival delays in pink) and the past arrival delays (values of train and validation tests), for the best model 1.

Similarly, Figure 17 presents the first $1,106$ forecasted values on the test set for the best model 2.

Figures 14, 15 imply that the best model 2 is associated with a smaller loss and MAE values. However, from Figures 16 and 17 one may say that best model 1 performs better on the unseen values (test set). After a rapid verification of the related code no technical error has been found. However, as this work is completed within too little time perhaps an undetected error may occur. If the code examination is correct, a first potential justification of this result could be that the selected metric MAE and/or loss function (Mean Square Error) is not adequate for this model. As previously mentioned, it would be appropriate to let the software evaluate many metrics simultaneously and also tune the loss function.

Observation of all the plots of this model implies that the considered number of epochs is too small. As explained before longer trainings are required. They are not currently considered due to time constraints and available computer resources.

For the next models, the test evaluation metrics will be presented as well the inferences for the first $1,106$ timesteps for each of the two best considered models will be presented. However, all the related plots will be included in a folder and be available to be examined separately.

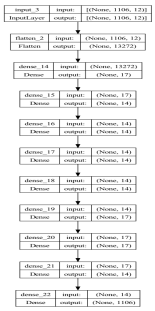

Figure 4: Graph Best Model 1, Dense Model 1.

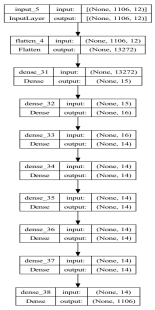

Figure 5: Graph Best Model 2, Dense Model 1.

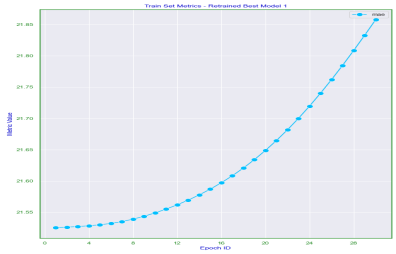

Figure 6: Train MAE when Retraining Best Model 1, Dense Model 1.

Remark 7.2.1 *The value* $1,106$ *employed for inferences and also as the size of the sequence length corresponds to the mean number of daily entries in the considered dataset after the data treatment. If a greater or smaller value of timesteps is wished*

Figure 7: Train MAE when Retraining Best Model 2, Dense Model 1.

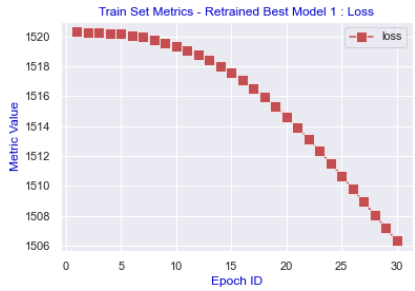

Figure 8: Train Loss Function when Retraining Best Model 1, Dense Model 1.

Figure 9: Train Loss Function when Retraining Best Model 2, Dense Model 1.

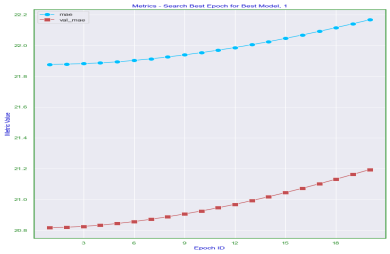

Figure 10: Train and Validation MAE when searching best number of epochs to retrain best model 1, Dense Model 1.

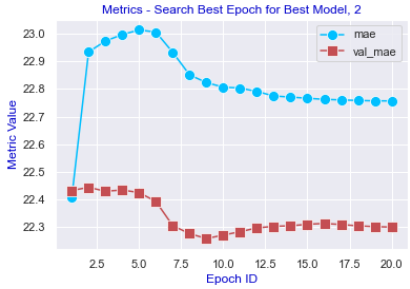

Figure 11: Train and Validation MAE when searching best number of epochs to retrain best model 2, Dense Model 1.

Figure 12: Train and Validation Loss when searching best number of epochs to retrain best model 1, Dense Model 1.

Figure 13: Train and Validation Loss when searching best number of epochs to retrain best model 2-Dense Model 1.

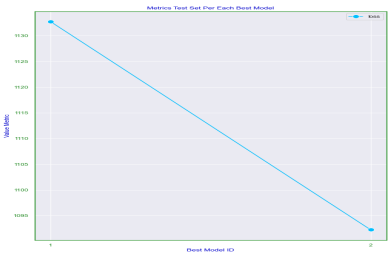

Figure 14: Loss value, Test Set, Best Models 1, 2, Dense Model 1.

Figure 15: MAE, Test Set, Best Models 1, 2, Dense Model 1.

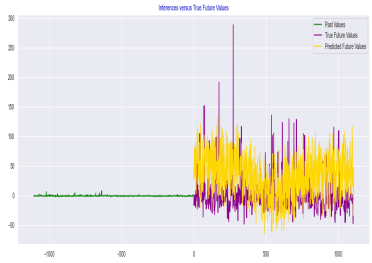

Figure 16: Inferences: 1, 106 timesteps, Best Model 1, Dense Model 1.

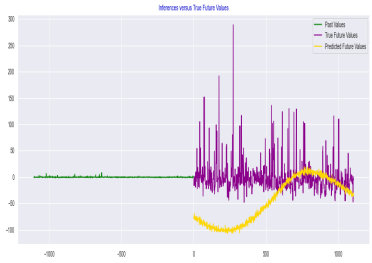

Figure 17: Inferences: 1, 106 timesteps, Best Model 2, Dense Model 1.

to be consider, this can be modified by selecting a different value for the variables val_nb_past_seq_lengths and/or val_nb_future_seq_lengths (current value equal to 1*).*

7.2.1 Experimenting Model Dense 1, shuffled-case, BayesianOptimization Tuner

A similar experimentation to the one described in § 7.2 is examined. The only difference is that sequences in batches are shuffled in order to use samples not necessarily close in time.

Figures 18,19 represent the graph of the two best models. The best model 1 is comprised of four hidden layers while best model 2 consists of six hidden layers.That is because the number of layers is determined by the tuner. For both models, the number of units regarding each layer is also selected by the tuner consequently this number varies from one model to another. Only the last dense layer, outputs the same number of units, for both models. This is required by the model in order to obtain an estimated value regarding flight arrival delays for each desired timestep.

The evaluation metrics on the test set are:

- the best model 1 is associated with a loss equal to 1150.49 and MAE equal to 21.8
- the best model 2 is associated with a loss equal to 1133.98 and MAE equal to 21.07.

The forecasted arrival delays for the first 1, 106 future timesteps are shown in Figures 20,21.

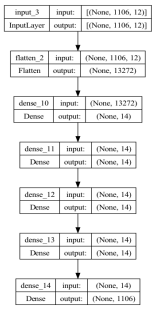

Figure 18: Graph Best Model 1, Dense Model 1, BayesianOptimisation tuner, Shuffled Entries.

7.3 Experimenting Model Dense 1-case Gridsearch Tuner

The model presented in § 7.2 is now exploited considering the GridSearch tuner (variable val_key_tuner_class values 3). This tuner will iterate over all possible hyperparameter combinations. However, as the maximum number of models to be examined is initialized to four, (input variable val_max_trials= 4) no more that four configurations of Dense model 1 will be created.

Figures 22,23 represent the graph of the two best models.

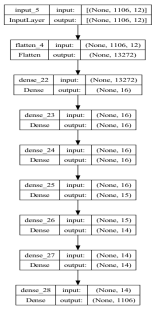

Figure 19: Graph Best Model 2, Dense Model 1, BayesianOptimisation tuner, Shuffled Entries.

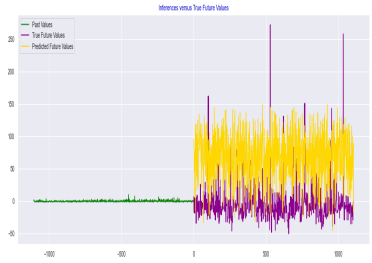

Figure 20: Inferences: $1,106$ timesteps, Best Model 1, Dense Model 1, BayesianOptimisation tuner, Shuffled Entries.

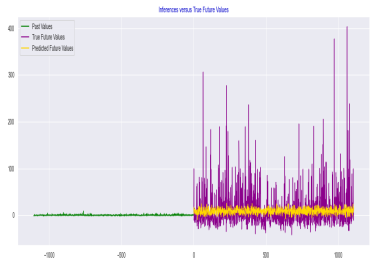

Figure 21: Inferences: $1,106$ timesteps, Best Model 2, Dense Model 1,BayesianOptimisation tuner, Shuffled Entries.

The evaluation metrics on the test set are:

- the best model 1 is associated with a loss equal to $1,078.843$ and MAE equal to 19.348
- the best model 2 is associated with a loss equal to $1,085.935$ and MAE equal to 19.586.

The forecasted arrival delays for the first $1,106$ future timesteps are shown in Figures 24,25.

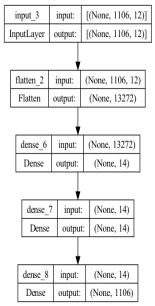

Figure 22: Graph Best Model 1, Dense Model 1, Gridsearch tuner.

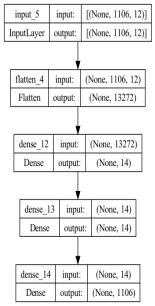

Figure 23: Graph Best Model 2, Dense Model 1, Gridsearch tuner.

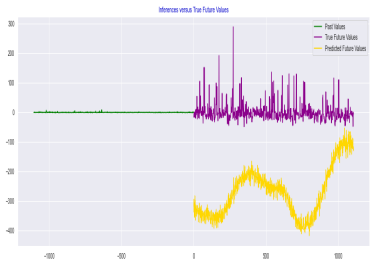

Figure 24: Inferences: $1,106$ timesteps, Best Model 1, Dense Model 1, Gridsearch.

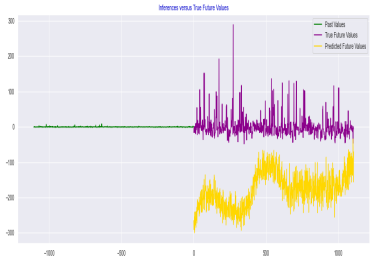

Figure 25: Inferences: $1,106$ timesteps, Best Model 2, Dense Model 1, Gridsearch.

7.4 Experimenting Model Dense 2-case BayesianOptimization Tuner

The model presented in § 2.2 is going to be considered.

The values of the input variables employed in the experimentation 7.2 are utilized. Figures 26,27 represent the graph of the best two models.

Regarding the test set the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1082.9229 and MAE equal to 19.637
- the best model 2 is associated with a loss equal to 1078.8546 and MAE equal to 19.3337.

The forecasted arrival delays for the first $1,106$ future timesteps are shown in Figures 28,29.

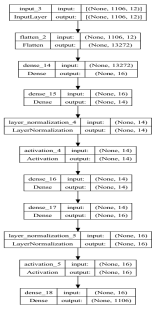

Figure 26: Graph Best Model 1, Dense Model 2.

7.5 Experimenting Model Dense 2-case Gridsearch Tuner

The model defined in § 7.4 is experimented under GridSearch tuner (val_key_tuner_class= 31). Figures 30,31 represent the graph of the best two models.

Regarding the test set the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1082.49 and MAE equal to 19.584

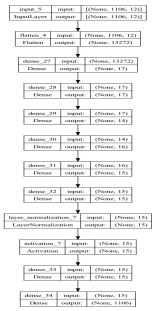

Figure 27: Graph Best Model 2, Dense Model 2.

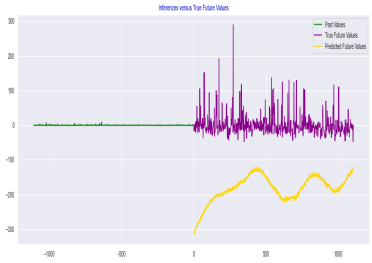

Figure 28: Inferences: $1,106$ timesteps, Best Model 1, Dense Model 2.

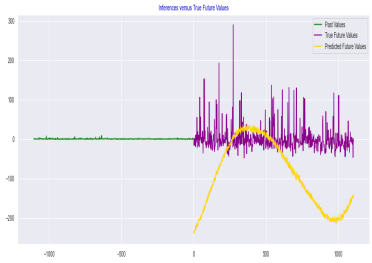

Figure 29: Inferences: $1,106$ timesteps, Best Model 2, Dense Model 2.

- the best model 2 is associated with a loss equal to 1092.225 and MAE equal to 19.809.

The forecasted arrival delays for the first 1, 106 future timesteps are shown in Figures 32,33.

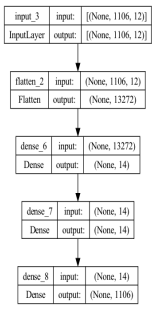

Figure 30: Graph Best Model 1, Dense Model 2, Gridsearch tuner.

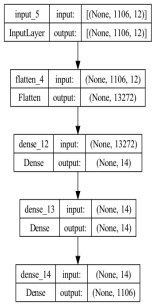

Figure 31: Graph Best Model 2, Dense Model 2, Gridsearch tuner.

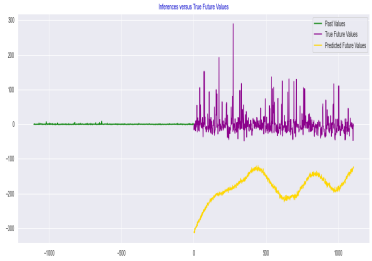

Figure 32: Inferences: 1, 106 timesteps, Best Model 1, Dense Model 2, Gridsearch.

7.6 Experimenting Model Dense 3-case BayesianOptimization Tuner

The model introduced in § 2.3 is going to be used.

Some of the input variables are:

1. val_min_nb_lay_model= 2

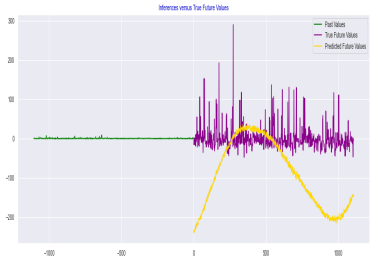

Figure 33: Inferences: $1,106$ timesteps, Best Model 2, Dense Model 2, Gridsearch.

2. val_max_nb_lay_model= 8
3. val_min_nb_units_model= 14
4. val_max_nb_units_model= 17
5. val_li_activ_fcts_model= $['relu','elu','selu']$
6. val_li_optimizers_model= $[“rmsprop”, “adam”, “adagrad”]$
7. val_max_trials= 4
8. val_epochs_tuner_search= 24
9. val_epochs_best_trained_model_search= 20
10. val_key_hypermodel_class= 3
11. val_key_tuner_class= 1

Figures 34,35 represent the graph of the best two models.
Regarding the test set the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1131.125 and MAE equal to 21.286
- the best model 2 is associated with a loss equal to 1119.0997 and MAE equal to 21.1821.

Figures 36,37 depict the forecasted arrival delays for the first $1,106$ future timesteps.

7.7 Experimenting Model Dense 3-case Gridsearch Tuner

The same model as in § 7.6 is here tried using the Gridsearch tuner (val_key_tuner_class= 3)

Figures 38,39 represent the graph of the best two models.
Regarding the test set the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1080.8996 and MAE equal to 19.542

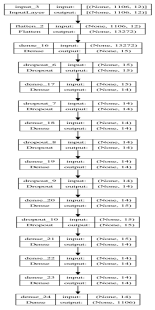

Figure 34: Graph Best Model 1, Dense Model 3.

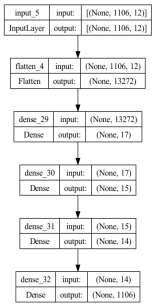

Figure 35: Graph Best Model 1, Dense Model 3.

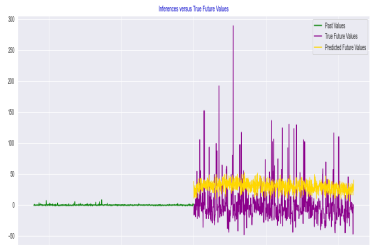

Figure 36: Inferences: 1, 106 timesteps, Best Model 1, Dense Model 3.

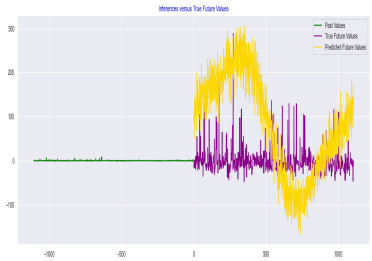

Figure 37: Inferences: 1, 106 timesteps, Best Model 2, Dense Model 3.

- the best model 2 is associated with a loss equal to 1098.1762 and MAE equal to 19.9466.

Figures 40,41 depict the forecasted arrival delays for the first $1,106$ future timesteps.

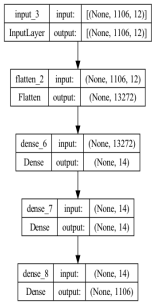

Figure 38: Graph Best Model 1, Dense Model 3, Gridsearch tuner.

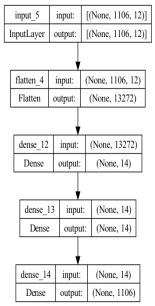

Figure 39: Graph Best Model 2, Dense Model 3, Gridsearch tuner.

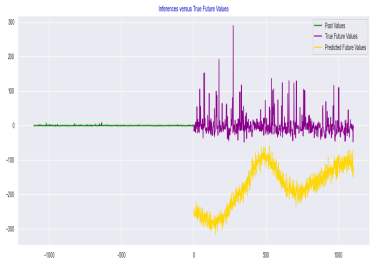

Figure 40: Inferences: $1,106$ timesteps, Best Model 1, Dense Model 3, Gridsearch.

7.8 Experimenting Model SimpleConv1D 1-case BayesianOptimization Tuner

It would be of interest to examine whether the considered dataset can be approached by the *translation invariance hypothesis*. Thus, the temporal convent topology introduced

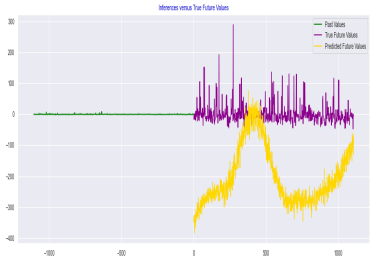

Figure 41: Inferences: $1,106$ timesteps, Best Model 2, Dense Model 3, Gridsearch.

in § 2.5 is examined aiming to find whether detecting the same representations over time and/or space could be useful. One could expect efficiency of this approach if there exists a timescale to be considered according to which the dataset presents a cycled behavior. Additionally, the existence of "Maxpooling" and "Global Average Pooling" layers in this model interferes in the data order, which may become a compromising element when dealing with spatiotemporal problems as here.

The three Conv1D layers of this model employ eight filters, are associated with a kernel size equal to twenty four, twelve and six (values for the first, second and third Conv1D layer respectively) and are activated using the "relu" activation function. The pool size of each following Maxpooling layer equals to two.

The loss function and the learning rate are tuned by the model. For simplicity (and not efficiency due to computer limitations), as explained in § 7.1 a single loss function is employed in this experiment ("mean square error").

Figures 42, 43 represent the graph of the two best models.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1140.336 and MAE equal to 21.97
- the best model 2 is associated with a loss equal to 1140.8499 and MAE equal to 21.41.

Figures 44,45 depict the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively.

Observation of Figure 44 shows that despite the error this model configuration tries to approximate the global tendency of the delay behavior. Observe how the yellow curve (estimated values) decreases over time. The pink plot (true values) shows occurence of significant pics in delays regarding the majority of delay values. In other words some particular flights present significantly higher values for the arrival delay regarding the remaining fights. Although these picks corresponding to increased delay values occur over all the considered timesteps their value tends to diminish. Roughly speaking, one could say that the behavior of the forecasted values (in yellow) reflects this tendency.

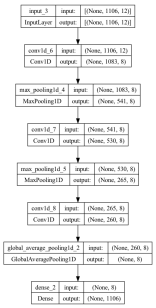

Figure 42: Graph Best Model 1, Model SimpleConv1D 1.

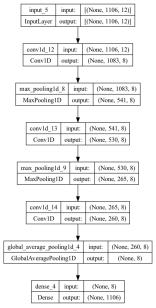

Figure 43: Graph Best Model 2, Model SimpleConv1D 1.

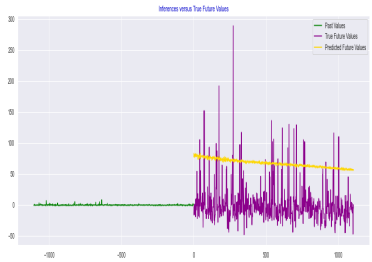

Figure 44: Inferences: 1, 106 timesteps, Best Model 1, Model SimpleConv1D 1.

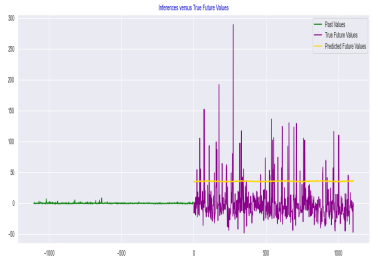

Figure 45: Inferences: 1, 106 timesteps, Best Model 2, Model SimpleConv1D 1.

7.9 Experimenting Model SimpleConv1D 2-case BayesianOptimization Tuner

This convent model tunes the number of layers, filters, kernel size and pool size. As other models it also selects the optimiser and the learning rate (even if "adam" algorithm is selected as optimizer and the loss.

The specific variables for this model are initialized with the following values:

1. val_min_nb_lay_model= 2
2. val_max_nb_lay_model= 4
3. val_min_nb_filters_conv1d= 6
4. val_max_nb_filters_conv1d= 10
5. val_min_nb_kernel_size_conv1d= 6
6. val_max_nb_kernel_size_conv1d= 24
7. val_step_nb_kernel_size_conv1d= 10
8. val_min_pool_size= 2
9. val_max_pool_size= 2
10. val_step_pool_size= $None$
11. val_li_activ_fcts_model= $[\text{“}relu'']$

With the aim of showing that larger number of trials is imposed, this model is experimented twice. Each of the two executions employs the same values for the input variables defining the desired ranges of hyperparameter values for the tuner to select.

Remark 7.9.1

- *The current version of this model may presents errors due to wrong values selected along the dimensioning, MAxpooling layers etc. As mentioned in § 2 a smarter version is required resulting to refined hyperparameter tuning.*
- *In the present experiment the same value for the "pool size " was desired to be employed for all convent layers. This is the reason for which the min and max values of the "pool size " take the same value.*
- *A single activation function,'relu" was considered in the list of the candidate ones. As mentioned before, the purpose fo the tuner is to try different possibilities and select the best possible one. Consequently, more options for activation functions should be examined.*

7.9.1 **Execution** 1

Figures 46, 47 represent the graph of the two best models.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1149.785 and MAE equal to 21.9025
- the best model 2 is associated with a loss equal to 1098.7955 and MAE equal to 20.226.

Figures 48,49 depict the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively.

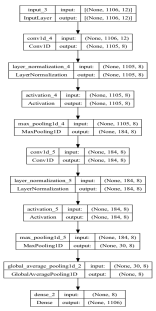

Figure 46: Graph Best Model 1, Model SimpleConv1D 2, 1st run.

Figure 47: Graph Best Model 1, Model SimpleConv1D 2, 1st run.

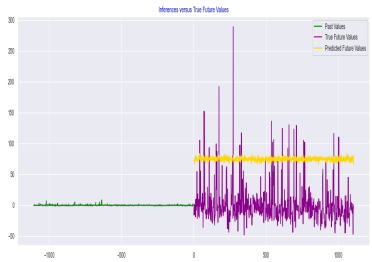

Figure 48: Inferences: $1,106$ timesteps, Best Model 1, SimpleConv1D 2, run 1.

Figure 49: Inferences: $1,106$ timesteps, Best Model 2, SimpleConv1D 2, run 1.

7.9.2 Execution 2

Figures 50, 51 represent the graph of the two best models during the second execution.
Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1155.086 and MAE equal to 22.055
- the best model 2 is associated with a loss equal to 1147.0189 and MAE equal to 21.7177.

Figures 52,53 depict the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively. One observes that now the two best models produce similar inferences. Observation of Figures 50, 51 shows the the architecture of the two best best models differs regarding the occurence of Layernormalization layers. For instance, in the best model 1 the first Conv1D layer is followed by a layer normalization and then an activation where this is not the case for the best model 2. Similarly, the second Conv1D layer of best model 2 is followed by a layer normalization while there is no normalization for the second convent layer of best model 1.

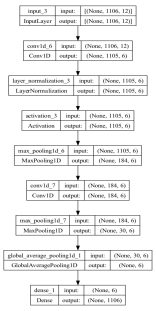

Figure 50: Graph Best Model 1, Model SimpleConv1D 2, 2nd run.

7.10 Experimenting Model RNN_model_1-case BayesianOptimization Tuner

The model presented in § 2.6 is going to be examined.

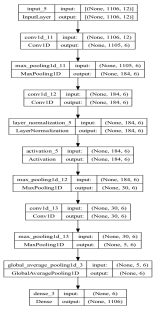

Figure 51: Graph Best Model 1, Model SimpleConv1D 2, 2nd run.

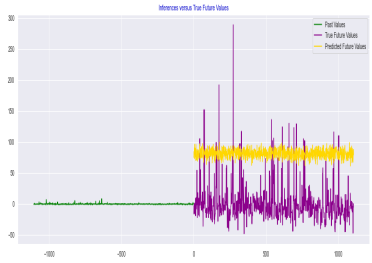

Figure 52: Inferences: 1, 106 timesteps, Best Model 1, SimpleConv1D 2, run 2.

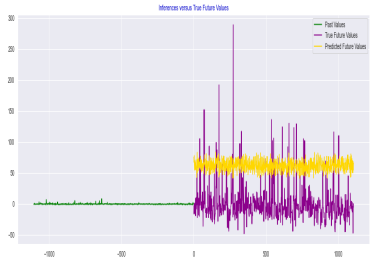

Figure 53: Inferences: 1, 106 timesteps, Best Model 2, SimpleConv1D 2, run 2.

As a spatiotemporal problem (timeseries sequences) is considered the event order should be respected. Furthermore, the occurence of one event may affect following ones. Consequently, the event order and causality should be taken into consideration. Contrary to Dense and Convent1D layers, recurrent layers maintain in memory the state of the system. Thus, at each timestep, the output is computed by combining the related input with the system state at the present timestep as well information of the system during previous timesteps.

Let's now see how a simple recurrent model behaves for the problem of flight delays estimation. As explained before, no definite conclusions should be taken out of any of the presented experiments since a very small part of the *hypothesis space* defined by each model is investigated (too few model configurations within the related hypothesis space are examined).

Figures 54, 55 represent the graph of the two best models. One observes that the two models are associated with the same architecture. This is an expected result as this model employs a fixed number of layers layers and units (single LSTM layer with fourteen units). It is only the choice of the optimizer, loss and the value of the learning rate that are decided by the BayesianOptimization tuner.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1132.72 and MAE equal to 21.01
- the best model 2 is associated with a loss equal to 1133.639 and MAE equal to 21.19.

Figures 56,57 illustrate the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively. Both plots show that the resulting inference curve (in yellow) for both best models one and two is trying to capture the general tendency of the true arrival delay evolution (in pink). When pics occur the forecasted values are slightly increased/decreased depending the pick direction. This observation holds true for all the $1,106$ future time steps.

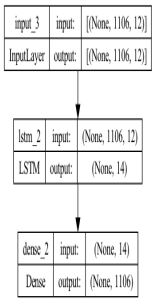

Figure 54: Graph Best Model 1, RNN_model_1.

7.10.1 Experimenting Model RNN_model_1, Shuffled-case, BayesianOptimization Tuner

A variation of the experimentation presented in § 7.10 is discussed. The same model and input values are employed with the difference now that train, validation and test

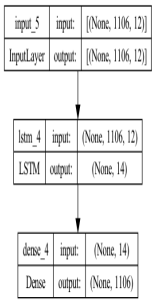

Figure 55: Graph Best Model 2, RNN_model_1.

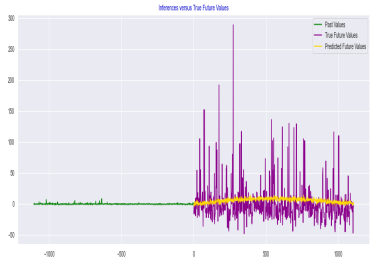

Figure 56: Inferences: $1,106$ timesteps, Best Model 1, RNN_model_1.

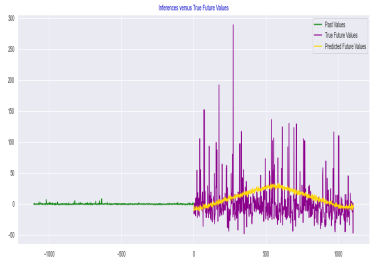

Figure 57: Inferences: $1,106$ timesteps, Best Model 2, RNN_model_1.

samples are now shuffled hoping to disturb the time order of consecutive sequences within the batches. Thus, entries will not necessarily be close in time. However, as short training duration and a limited number of hyperparameter configuration sets are tried, no comparison between the shuffled and non shuffled versions will be considered.

Figures 58, 59 illustrate the graph of the two best models. As expected, both models are associated with the same network architecture.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1150.498 and MAE equal to 21.8
- the best model 2 is associated with a loss equal to 1133.98 and MAE equal to 21.07.

Figures 60,61 illustrate the forecasted arrival delays for the first $1,106$ future (shuffled) timesteps for the best models 1 and 2 respectively.

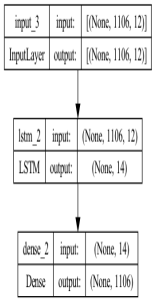

Figure 58: Graph Best Model 1, RNN_model_2, Shuffled Entries.

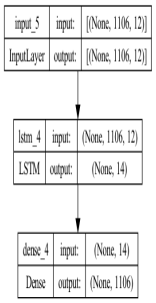

Figure 59: Graph Best Model 2, RNN_model_2, Shuffled Entries.

7.10.2 Experimenting Model RNN_model_1, Partially Shuffled-case, BayesianOptimization Tuner

Another variation of the experimentation presented in § 7.10 is considered. In this case, aiming at disordering consecutive sequences within the train and validation tests, the involved samples are shuffled. Thus, entries will not necessarily be close in time.

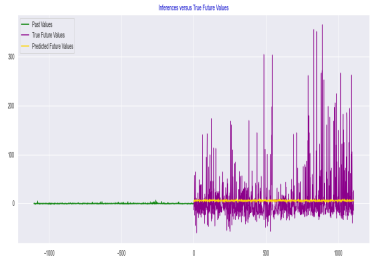

Figure 60: Inferences: $1,106$ timesteps, Best Model 1, RNN_model_2, Shuffled Entries.

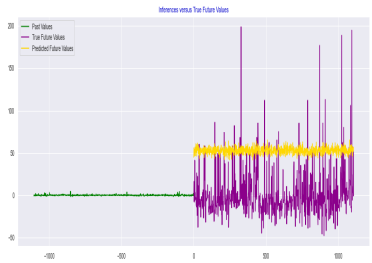

Figure 61: Inferences: $1,106$ timesteps, Best Model 2, RNN_model_2, Shuffled Entries.

However, as short training duration and a limited number of hyperparameter configuration sets are tried, no comparison between the shuffled and non shuffled versions will be considered.

Remark 7.10.1

- *Within the experimentation in § 7.10.2 the batches of sequences for the train and validation datasets (including the targets) are shuffled. However, the test sets are time ordered. However one may consider different combinations of shuffled datasets (e.g. shuffle only the train sets etc.).*

- *As discussed before, within the presented experiments, each considered data sequence is comprised of the* $1,106$ *entries (this is the mean number of daily observations for the considered dataframe). Thus, each data sequence is not necessarily comprised of distinct timesteps as it may exist flights departing the same or distinct airports associated with the same departure time. Based on this information, one can be inspired to shuffle only the considered entries set while leaving the targets sets intact.*

Figures 62, 63 illustrate the graph of the two best models. Since this model does not tune the number of layers and the involved units, both best models are associated with the same network architecture.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1154.6 and MAE equal to 22
- the best model 2 is associated with a loss equal to 1159.6 and MAE equal to 22.2.

Figures 64,65 illustrate the forecasted arrival delays for the first $1,106$ future (shuffled) timesteps for the best models 1 and 2 respectively. One may notice that both best models provide close forecasted values. This is a reasonable observation as each possible configuration of this model consists of distinct values for the optimizer and the learning rate.

Hence four different model configurations are tried.

- **Model Configuration** 1: optimizer: adam, learning rate: 0.00028636418591913084, Score: 21.76
- **Model Configuration** 2: optimizer: adagrad, learning rate: 0.0014196495433366004, Score: 20.82
- **Model Configuration** 3: optimizer: adam, learning rate: 0.0005144475412729949, Score: 22.15
- **Model Configuration** 4: optimizer: rmsprop, learning rate: 0.00013785668232897005, Score: 21.06.

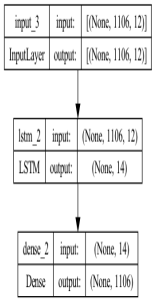

Figure 62: Graph Best Model 1, RNN_model_2, Partially Shuffled Entries.

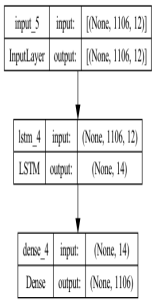

Figure 63: Graph Best Model 2, RNN_model_2, Partially Shuffled Entries.

Figure 64: Inferences: $1,106$ timesteps, Best Model 1, RNN_model_2.

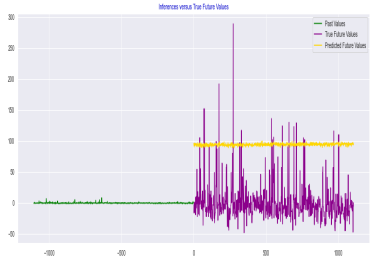

Figure 65: Inferences: $1,106$ timesteps, Best Model 2, RNN_model_2.

7.11 Experimenting Model RNN_model_2-case BayesianOptimization Tuner

The recurrent model described in § 2.7 is experimented. This model employs two GRU layers with thirty two units each associated with a recurrent dropout rate equal to 0.25.

The employed values for the input variables are:

1. val_li_activ_fcts_model= $[“tanh”]$
2. val_li_optimizers_model= $[“rmsprop”, “adam”, “adagrad”]$
3. val_max_trials= 4
4. val_epochs_tuner_search= 24
5. val_epochs_best_trained_model_search= 20
6. val_key_hypermodel_class= 10
7. val_key_tuner_class= 1 (BayesianOptimization tuner).

Figures 66, 67 illustrate the graph of the two best models. As expected, both models are associated with the same network architecture.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1084.073 and MAE equal to 19.54

- the best model 2 is associated with a loss equal to 1132.253 and MAE equal to 21.05.

Figures 68,69 illustrate the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively.

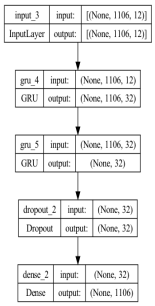

Figure 66: Graph Best Model 1, RNN_model_2.

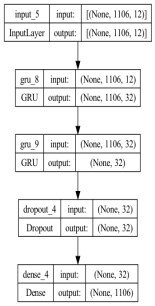

Figure 67: Graph Best Model 2, RNN_model_2.

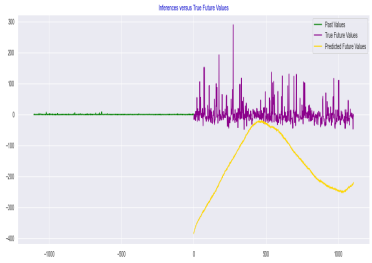

Figure 68: Inferences: $1,106$ timesteps, Best Model 1, RNN_model_2.

7.12 Experimenting Model RNN_model_3-case BayesianOptimization Tuner

The recurrent model described in § 2.8 is experimented. This model examines whether to use LayerNoralization layers. Furthermore, it tunes: the number of layers and involved units, the layer activation function, the dropout rate and recurrent dropout rate

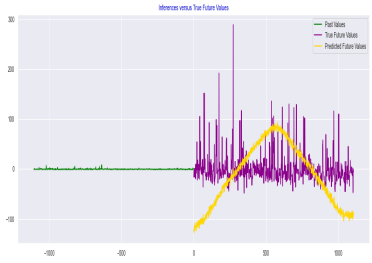

Figure 69: Inferences: $1,106$ timesteps, Best Model 2, RNN_model_2.

as well the optimizer and the involved learning rate, loss and evaluation metrics a This model employs LSTM layers.

The employed values for the input variables are:

1. val_min_nb_lay_model= 1
2. val_max_nb_lay_model= 4
3. val_min_nb_units_model= 18
4. val_max_nb_units_model= 20
5. val_li_activ_fcts_model= $[“tanh”, “relu”]$
6. val_li_optimizers_model= $[“rmsprop”, “adam”, “adagrad”]$
7. val_max_trials= 2
8. val_epochs_tuner_search= 24
9. val_epochs_best_trained_model_search= 20

Figures 70, 71 depict the graph of the two best models. Rather different topologies are defined by each model as a significant difference one the number of employed layers by each model is observed (four LSTM layers are employed by best model 1 and two LSTM layers by best model 2). For the best model 1, a single LayerNormalization layer occurs following the second LSTM layer while a single LayerNormalization layer also occurs for best model 2 just after the first LSTM layer.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1128.909 and MAE equal to 21.015
- the best model 2 is associated with a loss equal to 1113.621 and MAE equal to 20.99.

Figures 72,73 present the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively. One sees that for the best model 1 the forecasted values try to approach the behavior of true values although there is a significant marge between estimated and true values.

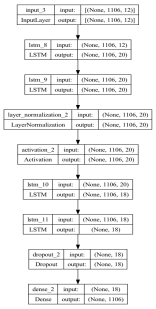

Figure 70: Graph Best Model 1, RNN_model_3.

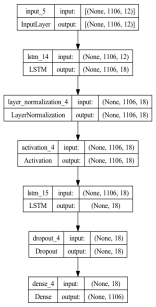

Figure 71: Graph Best Model 2, RNN_model_3.

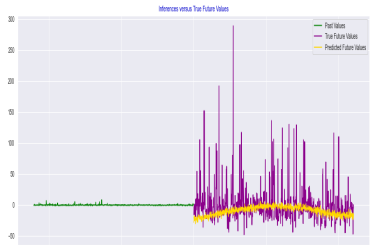

Figure 72: Inferences: 1, 106 timesteps, Best Model 1, RNN_model_3.

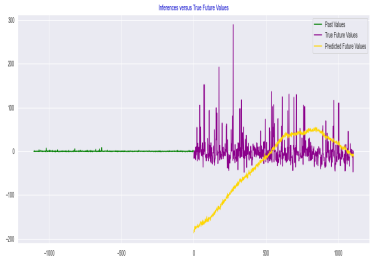

Figure 73: Inferences: 1, 106 timesteps, Best Model 2, RNN_model_3.

7.13 Experimenting Model RNN_model_4-case BayesianOptimization Tuner

The recurrent bidirectional model employing LSTM type of layers described in § 2.9 is experimented. RNN models process ordered data sequences. The interest of the bidirectional layer is to investigate whether a reversed order may imply more performant data representations.

The employed values for the input variables to the model are:

1. val_min_nb_lay_model= 1
2. val_max_nb_lay_model= 4
3. val_min_nb_units_model= 34
4. val_max_nb_units_model= 44
5. val_li_activ_fcts_model= [“$tanh''$]
6. val_li_optimizers_model= [“$rmsprop''$, “$adam''$, “$adagrad''$]
7. val_max_trials= 2
8. val_epochs_tuner_search= 24
9. val_epochs_best_trained_model_search= 20
10. val_key_hypermodel_class= 9
11. val_key_tuner_class= 1 (BayesianOptimization tuner).

Due to increased potential number of units per layer, varying between 34 and 44, long trainings are required for this model (not applied in the present experimentations as limited computer ressources and time constraints are available).

Figures 74, 75 depict the graph of the two best models. Both models employ two bidirectional LSTM layers associated with different number of units (80, 70) and recurrent dropout. As planned a dropout layer before the last (Dense type) layer is employed.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1135.25 and MAE equal to 21.18
- the best model 2 is associated with a loss equal to 1082.11 and MAE equal to 19.5.

Figures 76,77 present the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively.

Remark 7.13.1

When different activation functions than “tanh” were employed in this model there was a data dimensioning problem prohibiting the program execution.

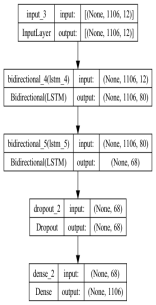

Figure 74: Graph Best Model 1, RNN_model_4.

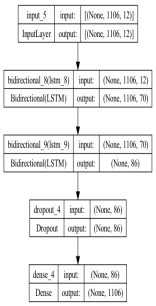

Figure 75: Graph Best Model 2, RNN_model_4.

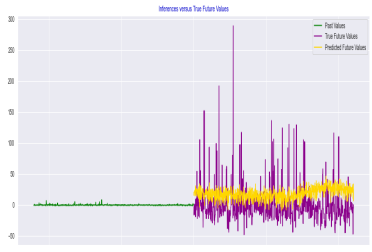

Figure 76: Inferences: 1, 106 timesteps, Best Model 1, RNN_model_4.

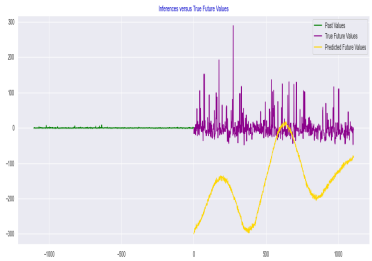

Figure 77: Inferences: 1, 106 timesteps, Best Model 2, RNN_model_4.

7.14 Experimenting Model RNN_model_5-case BayesianOptimization Tuner

The recurrent model described in § 2.10 is experimented. This model employs bidirectional GRU layers.

The employed values for the input variables are:

1. val_min_nb_lay_model= 1
2. val_max_nb_lay_model= 2
3. val_min_nb_units_model= 14
4. val_max_nb_units_model= 17
5. val_li_activ_fcts_model= [“$tanh''$, “$relu''$, “elu'', “$selu''$]
6. val_li_optimizers_model= [“$rmsprop''$, “$adam''$, “$adagrad''$]
7. val_max_trials= 4
8. val_epochs_tuner_search= 24
9. val_epochs_best_trained_model_search= 20
10. val_key_hypermodel_class= 10
11. val_key_tuner_class= 1 (BayesianOptimization tuner).

Figures 78, 79 depict the graph of the two best models. Both models employ two bidirectional GRU layers which is the max allowed number of layer for this model configuration associated with different number of units.

Regarding the test set, the evaluation metrics are:

- the best model 1 is associated with a loss equal to 1144.875 and MAE equal to 21.71
- the best model 2 is associated with a loss equal to 1076.299 and MAE equal to 19.42.

Figures 80,81 present the forecasted arrival delays for the first $1,106$ future timesteps for the best models 1 and 2 respectively.

7.15 Arguing the Presented Experiments

As previously discussed, no definite conclusions can be drawn or comparative studies should be considered regarding the results of the presented DNN models. That is because, a minimal search of possible solutions within the *hypothesis space* is realized due to computer and time constraints. In other words, too few hyperparameter combinations are tried while a short learning phase is imposed prohibiting the model convergence towards a reduced error.

Nevertheless, regarding the first approach where the model learning is made from a single dataframe involving all possible values of he origin airports, the minimum

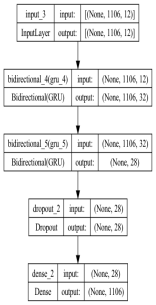

Figure 78: Graph Best Model 1, RNN_model_5.

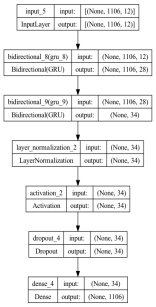

Figure 79: Graph Best Model 2, RNN_model_5.

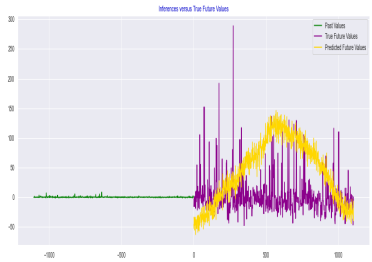

Figure 80: Inferences: 1, 106 timesteps, Best Model 1, RNN_model_5.

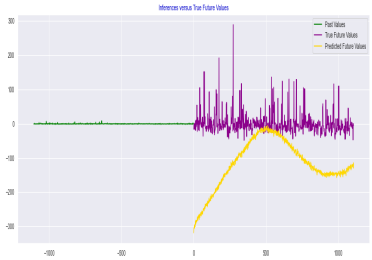

Figure 81: Inferences: 1, 106 timesteps, Best Model 2, RNN_model_5.

value of the considered evaluation metric (*mean absolute error*) on the test set is 19.42 while the max observed value of the error equals 22.2. This max value is observed by a single model while for all other models the max value of the error is around 21. Regarding, the baseline model see § 6 the min, max value of the error in the test set are equal to 22.3 and 23 respectively. As one sees, even with poorly trained DNN models potentially involving a non adequate architecture improved responses are provided.

8 Model Experimentation- Case Learning Single Origine

In this section each model will forecast the arrival delays of flights departing from the same "ORIGIN" airport and where the model training phase will involve only flights associated with this "ORIGIN". Hence, for each possible value of feature "ORIGIN", distinct subsets of data are created each one associated with entries for a given "ORIGIN". Each desired model will make assumptions on the data involved in a single data subset and use the available information in association with these assumptions to learn how to forecast the forthcoming arrival delays regarding flights departing from the associated origin airport. Thus, each model will learn to inference flight arrival delays from a given origin independently from the other origins.

Problem Statement

The hypothesis of *independence* between the distinct values of feature "ORIGIN" is going to be explored. Hence, a dataset involving at least a minimum number of n_e entries all associated with the same value for feature "ORIGIN" is considered. Using information on the past n_p entries for flights departing the same airport (with $n_p < n_e$), the arrival delays for the future n_f flights (departing the same airport) are aimed to be forecasted. Similar to the study presented in § 7, involving all values of feature "ORIGIN" during the training phase of each model, all model experimentations are learning from past information equal to the mean number of daily flight departures (what is also called observations in this work) to estimate the arrival delays of the same number of forthcoming flight departures ($n_p = n_f =$ mean number of observations or flight departures from a given origin, per day).

Nevertheless, as values of variables n_p and n_f are selected by the user one may examine different learning and/or forecasting periods.

In what follows a few models will be experimented. As explained before due to limited trials and training duration the resulting numbers should not be used for definite conclusions.

For each model, results for the following origin airports are available: IAD, SEA, LAX, ORD, TPA, DCA, LGA, OAK, BDL, PDX, BIL, BOS, PHL, LAS, DTW, DEN, OMA, RDU, SFO, ONT, SMF, MSP, GEG, PHX, MCO, MSY, RNO, JFK, BUF, DSM, SAN, MCI, MDT, CLE, OKC, PVD, FSD, IAH, EWR, BWI, BTV, ROC, ICT, MIA, RIC, TUL, HNL, DFW, JAC, PIT, AUS, SNA, SJC, BOI, ALB, DAY, GRR, CMH, HDN, SLC, SJU, EGE, ABQ, SAT, ATL, CLT, RSW, MHT, KOA, OGG, LIH, PSP, STL, TUS, STT, IND, JAX, CVG, BZN, RAP.

Hereafter, a few results for two important airports are presented: LAX standing for Los Angeles International Airport and EWR standing for Liberty International Airport, Newark, New Jersey.

8.1 Experimenting Model Dense 1-case BayesianOptimization Tuner

The model presented in § 2.1 is going to be considered. The same values for the input variables as the experiment 7.2 are utilized except of the sequence length for the training, validation and test sets. This value still remains the mean number of observations per day which now depends upon the "ORIGIN" airport (a priori each airport has its own number of daly flight departures. These values are 77 and 13 daily domestic flights departing LAX, EWR respectively, regarding UA airlines).

Figures 82, 83 illustrate the correlation coefficient matrices for the datasets of LAX and EWR airports respectively.

Figures 84,85 represent the graph of the best two models for the LAX airport where Figures 86,87 represent the graph of the best two models for the EWR airport. Observe how the best two models for the LAX airport correspond to a different system dimensioning (different number of layers and units). However, both models output 77 units as this is the mean number of observations (departures) per day from the LAX airport for UA (recall that in the presented experiments the window of forecasts equals to the mean value of departures per day). A similar assessment can be stated for the EWR airport where now the last Dense layer outputs 13 units, the mean number of daily domestic flights operated by UA departing from EWR airport.

Figures 88,89 represent the train MAE for the two retrained best models regarding the LAX airport where Figures 90,91 illustrate the train MAE for the two retrained best models regarding the EWR airport.

Observe that the best models 1 and 2 for LAX airport are retrained for 3 and 4 epochs respectively. When the "best models are determined then the best number of epochs to retrain each best model is also computed. Each best model is retrained for a little longer than the best number of epochs as it is comprised of both train and validation datasets. As one sees a small number of epochs is required. However no conclusions should be taken as small train duration is considered due to computer limitations (value defined by input variable val_epochs_tuner_search in file cc_fi_input_variables).

A similar situation holds true for the train MAE of the two best models during their retraining regarding EWR airport. Both models require a longer retraining than the one for the LAX airport (with a very small difference regarding the number of retraining epochs between best models 1 and 2).

Figures 92,93 represent the train loss for the two retrained best models regarding the LAX airport where Figures 94,95 illustrate the train loss for the two retrained best models regarding the EWR airport.

The forecasted arrival delays for the first 77 future flight departures (not necessarily coinciding with distinct timesteps) for LAX airport are depicted in Figures 96,97. Similarly, Figures 98,99 show the inferences for the first 13 future departures from EWR airport regarding domestic flights of UA.

In each of these four figures, the green curve plots the past values, the pink line corresponds to the true future values (true future arrival delays) while the yellow one corresponds to the forecasts.

For the currently employed values (aiming only at experimenting the functioning of the proposed work) one sees that regarding LAX airport, best model 1 manages to provide better inferences than those of model 2. Regarding origin EWR inferences

of both best models are “distant” from the true values for all the considered future forecasts.

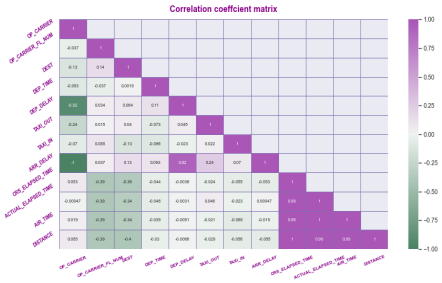

Figure 82: Correlation Coefficient Matrix, LAX.

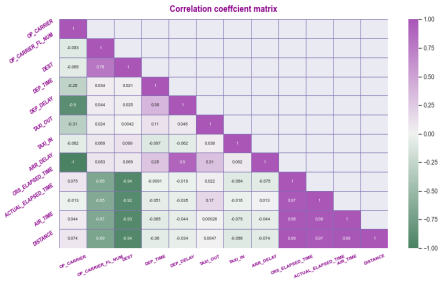

Figure 83: Correlation Coefficient Matrix, EWR.

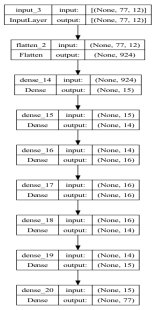

Figure 84: Graph Best Model 1, Dense Model 1, LAX.

8.2 Experimenting Model Dense 2-case BayesianOptimization Tuner

The model presented in § 2.2 is going to be experimented.

Figures 100,101 represent the graph of the best two models for the LAX airport where Figures 102,103 represent the graph of the best two models for the EWR airport.

As in § 8.1, here too the two best models for LAX airport define a different topology involving different number of layers and units as well LayerNormalization type of layers. A similar observation can be stated for EWR airport too.

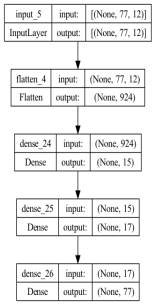

Figure 85: Graph Best Model 2, Dense Model 1, LAX.

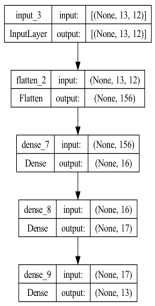

Figure 86: Graph Best Model 1, Dense Model 1, EWR.

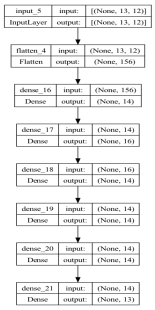

Figure 87: Graph Best Model 2, Dense Model 1, EWR.

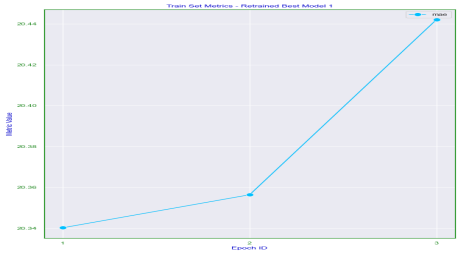

Figure 88: Train MAE, Retrained Best Model 1, Dense Model 1, LAX.

Figure 89: Train MAE, Retrained Best Model 2, Dense Model 1, LAX.

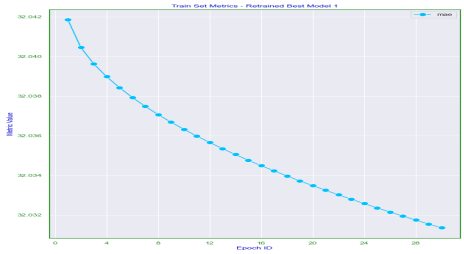

Figure 90: Train MAE, Retrained Best Model 1, Dense Model 1, EWR.

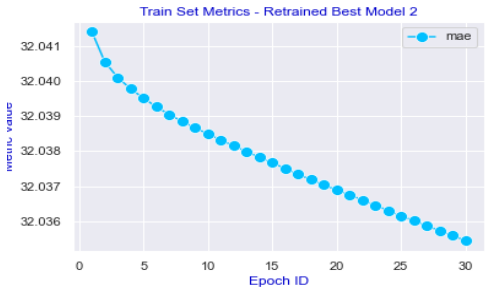

Figure 91: Train MAE, Retrained Best Model 2, Dense Model 1, EWR.

Figure 92: Train Loss, Retrained Best Model 1, Dense Model 1, LAX.

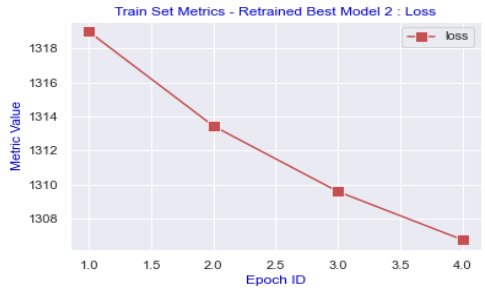

Figure 93: Train Loss, Retrained Best Model 2, Dense Model 1, LAX.

Figure 94: Train Loss, Retrained Best Model 1, Dense Model 1, EWR.

Figure 95: Train Loss, Retrained Best Model 2, Dense Model 1, EWR.

Figure 96: Inferences: 77 timesteps, Best Model 1, Dense Model 1, LAX.

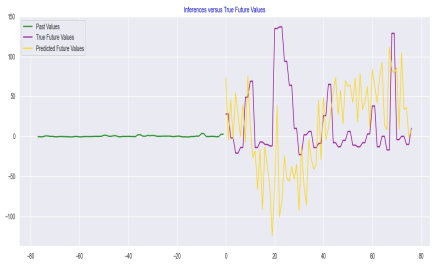

Figure 97: Inferences: 77 timesteps, Best Model 2, Dense Model 1, LAX.

Figure 98: Inferences: 13 timesteps, Best Model 1, Dense Model 1, EWR.

Figure 99: Inferences: 13 timesteps, Best Model 2, Dense Model 1, EWR.

Figures 104,105,106,107 ilustrate the inferences for the 77 and 13 first future flight departures from LAX, EWR airports respectively. Regarding inferences related to EWR airport one sees that the best model 1 for Dense model 2 is slightly improved regarding the best model 1 for Dense model 1 (see Figures 98 , 106).

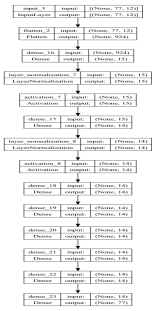

Figure 100: Graph Best Model 1, Dense Model 2, LAX.

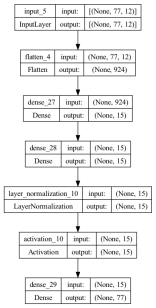

Figure 101: Graph Best Model 2, Dense Model 2, LAX.

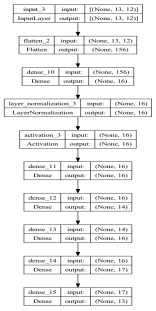

Figure 102: Graph Best Model 1, Dense Model 2, EWR.

8.3 Experimenting Model Dense 3-case BayesianOptimization Tuner

The model presented in § 2.3 is going to be experimented.

Figures 108,109 represent the graph of the best two models for the LAX airport where Figures 110,111 represent the graph of the best two models for the EWR airport.

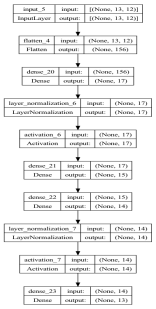

Figure 103: Graph Best Model 2, Dense Model 2, EWR.

Figure 104: Inferences: 77 timesteps, Best Model 1, Dense Model 2, LAX.

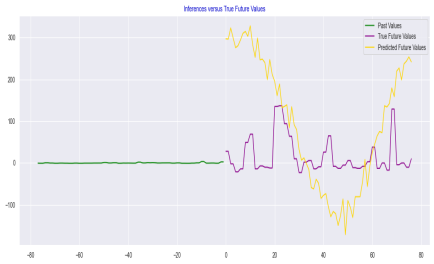

Figure 105: Inferences: 77 timesteps, Best Model 2, Dense Model 2, LAX.

Figure 106: Inferences: 13 timesteps, Best Model 1, Dense Model 2, EWR.

Figure 107: Inferences: 13 timesteps, Best Model 2, Dense Model 2, EWR.

For the case of LAX, the two selected best models are comprised of the same number of Dense hidden layers and Dropout layers. The topology of each best model is involving eight Dense hidden layers and two Dropout layers (in order to prevent from overfitting). However, each model applies Dropout layers at different levels. Hence, best model 1 applies Dropout after the second and third Dense layers. Regarding the best model 2, the 5th and 6th Dense layers are followed by DropOut layers. A similar observation holds true for the EWR origin where the best model 1 employs dropout after the first and second Dense layers while the best model 2 employs dropout after the first, third, fifth and seventh Dense hidden layers.

Figures 112,113,114,115 illustrate the inferences for the 77 and 13 first future flight departures from LAX, EWR airports respectively.

As Figures 114, 115 depict, regarding EWR, the best models 1 and 2 (case Dense model 3) still poorly perform during all the first thirteen inferences.

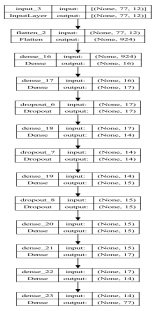

Figure 108: Graph Best Model 1, Dense Model 3, LAX.

8.4 Experimenting RNN model 1-case BayesianOptimization Tuner

The model presented in § 2.6 is going to be experimented.

Figures 116,117 represent the graph of the best two models for the LAX airport where Figures 118,119 represent the graph of the best two models for the EWR airport. One observes that all models have the same topology which is an expected result. Model RNN 1 is comprised of a fixed number of layers and units, employs the "tanh" activation function. This model tunes only the optimiser, the related learning rate (even in case of "adam" optimiser which adapts the learning rate for each weight of

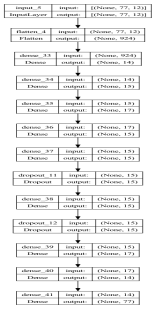

Figure 109: Graph Best Model 2, Dense Model 3, LAX.

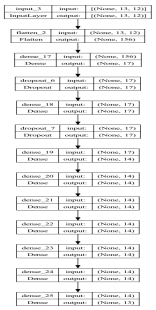

Figure 110: Graph Best Model 1, Dense Model 3, EWR.

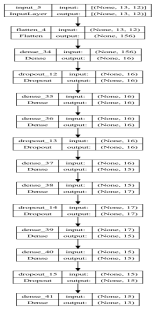

Figure 111: Graph Best Model 2, Dense Model 3, EWR.

Figure 112: Inferences: 77 timesteps, Best Model 1, Dense Model 3, LAX.

Figure 113: Inferences: 77 timesteps, Best Model 2, Dense Model 3, LAX.

Figure 114: Inferences: 13 timesteps, Best Model 1, Dense Model 3, EWR.

Figure 115: Inferences: 13 timesteps, Best Model 2, Dense Model 3, EWR.

the neural network) and the loss function.

Figures 120,121,122,123 illustrate the inferences for the 77 and 13 first future flight departures from LAX, EWR airports respectively. One sees that especially for EWR and for both best models poor performance results. However, it worths to be accentuated that with the current values of the input variables the model is not trained long enough and properly not with adequate hyperparameter values. Consequently, no conclusions or interpretations should be taken out of the presented results.

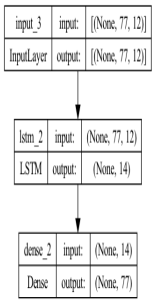

Figure 116: Graph Best Model 1, RNN Model 1, LAX.

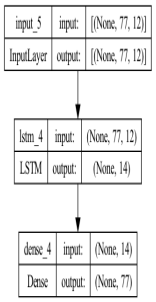

Figure 117: Graph Best Model 2, RNN Model 1, LAX.

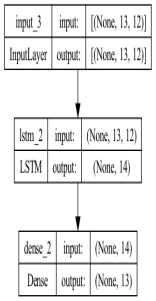

Figure 118: Graph Best Model 1, RNN Model 1, EWR.

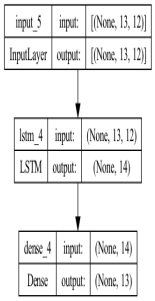

Figure 119: Graph Best Model 2, RNN Model 1, EWR.

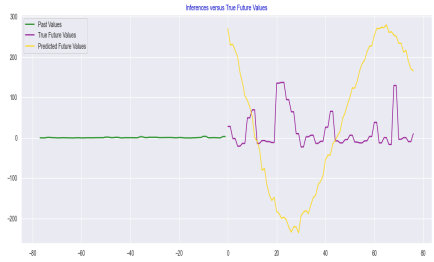

Figure 120: Inferences: 77 timesteps, Best Model 1, RNN Model 1, LAX.

Figure 121: Inferences: 77 timesteps, Best Model 2, RNN Model 1, LAX.

Figure 122: Inferences: 13 timesteps, Best Model 1, RNN Model 1, EWR.

Figure 123: Inferences: 13 timesteps, Best Model 2, Dense Model 3, EWR.

9 Discussion

Aiming at comparative studies when dealing with timeseries regression problems, a first attempt creating an accurate tool able to easily incorporate and evaluate various Deep Neural Network (DNN) models is introduced. An extensible library of various *adjustable models* is created where each model is associated with automated processes aiming to return the most efficient *versions* of it.

The design of a DNN model is of major importance as this defines the network topology, the assumptions made on the data from where the search of possible solutions is going to start. Choosing the appropriate network topology implies adequate constraints to the hypothesis space within which the model parameters (weights) will be determined. Evidently, the model architecture plays a crucial role. Association of theory and experiments can lead to efficient mappings between input and output data through models presenting good generalization, able to deal with original situations and hence able to provide competitive estimations.

Defining the design and hyperparameter values of any DNN model can be a challenging and time consuming task. Using *automation* whenever this is possible can significantly contribute to the achievement of competitive models while potentially can ease the user involvement. Autokeras [32] provides libraries for automated machine learning but for a fully automated model is still required a lot of work for being operational. Hyper-parametrized models where the network dimension and topology are defined through sets of hyperparameters can be a key element towards automated operations examining model performance.

Within this work various models are proposed offering different possibilities for the related hypothesis space. Each one involves different types of layers (with/without memory such as *Fully Connected*, *Recurrent* layers, translation invariant such as Conv1D layers). Models involving options preventing overfitting are proposed through the employment of adequate layers (*Dropout* layers, *dropout* rates, *Recurrent Dropout* layers and *recurrent dropout* rates). Possibilities allowing all neurons of a particular layer to follow the same distribution for all features of a given entry are also considered through LayerNormalization layers. Furthermore, additional hyperparameters are involved determining the model training duration (number of epochs), considered sizes for subsets of entries (batch size), sizes for each element of a batch (sequence length expressing the timesteps to consider during each sequential learning), choice of the optimizer (algorithm to employ in order to update the network), the loss function to

be optimized and the evaluation metrics through which the model performance will be appraised.

In order to experiment one of the suggested models and find optimized topologies one has only to define sets of desired values (or ranges of values) within which the involved model procedures will select (series of) hyperparameter values to examine. Different model configurations (different data assumptions) will be experimented automatically and among these trials the best performant model(s) will be returned as well the related estimations and the associated metric evaluations.

A first experimentation of each proposed model is presented. From the provided results no definite conclusions should be drawn as a limited number of trials and a very small sets for hyperparameter values are considered due to imposed constraints from the limited power of the employed computer and time shortage. However, the proposed methodology and evaluation tool remain valid for deeper and richer experiments.

10 Study Extend

A first approach regarding the creation of automated model architectures for time-series regression problems was previously discussed. Moreover, a tool appraising the performance of each model was also introduced. However, multiple objectives are remaining to be developed in order to achieve a competent automated software involving efficient models for trustworthy estimations.

In what follows some indications for potential future work are discussed.

1. Regarding the introduced models some proposals for further work are presented in the following items.

    - Increase the number of trials (hyperparameter value combinations) so as the model will exploit larger *areas* of the hypothesis space.
    - Additionally, the number of executions per trial should be extended. In order to control variance, each model configuration, according to a selected set of hyperparameter values, is experimented for a desired number of times (and take the average value). At present, for a given model each considered hyperparameter set is evaluated two times.
    - Vary the ranges of the hyperparameter values in order to allow larger, smaller and potentially more refined exploitations of the hypothesis space (min, max number of layers, units, dropout or recurrent rate, kernel size, pool size).
    - Create smarter constraints allowing or prohibiting the creation of specific types of layers such as LayerNormalization, DropOut layers.
    - Try different Keras tuners. The current models employ the BayesianOptimisation (Gaussian process) and/or GridSearch Tuner iterating over all possible hyperparameter values combination. However, other tuners are also available such as Hyperband tuner, Sklearn tuner (for Scikit-learn models) etc.
    - Employ feature engineering in some inputs (e.g. Departure Time) aiming at increasing the data efficiency.

- Consideration of different sequence lengths. At present the sequence lengths for all training, validation and test sets are determined by formula $s_l = m \times \alpha$ where s_l represent the sequence length, m is the mean number of daly observations and α is a positive integer indicating how far in the past or future, the system should go during the learning or inferencing phases respectively. The presented experimentations are using $\alpha = 1$ for both learning and forecasting. However, different values of α need to be experimented. Moreover, each entry does not necessarily represents a different timestep as at a given instant multiple flight departs may occur especially in the case when all values of feature "ORIGIN" are employed simultaneously during the training, inferencing period. One may define adequate sequence lengths such that each data sequence will involve the same number of timesteps. This would result to a more precise study especially when longer future predictions are desired or training considering multiple past information (exceeding the mean value of past observations).
- Consideration of larger sets of data. At present, information for year 2009 is only employed although datasets from 2009 to 2018 are available. It would be of interest to integrate more years in this study (and potentially try to determine cycles within a number of years. In other words, try to examine if there exists a *number of years* such that the arrival flight delays comportment approaches a repetitive scheme). Equivalently, involvement of more than one airlines should be considered. At present, data from UA airline is only exploited. Often, delays on a particular flight may generate future delays to other flights as well. If a particular airline is associated with increased delays then a model may detect the related influence of this company to the other flights (correspondances, airport management etc.).

2. Regarding new models, further ideas are discussed hereafter.

   - Conception of a model taking adaptive decisions regarding the network dimensioning (number layers, units etc.). At present, the values defining the limits (inferior, superior) of the search range should be defined by the user and the tuner selects a value within an interval the limits of which are defined by the min and max values. It would be of interest, to create an automated model able to to determine these limits and through learning update these values. Similarly for the learning rate, dropout rate, recurrent dropout, kernel size, pool size (whenever such hyperparameters apply). This part was one of the initial goals of this work which unfortunately is not presented here as it is not completed by today.
   - Adaptive models initializing single valued parameters should also be considered. For instance, in the presented work the number of training epochs should be defined by the user. When performance is not improved for a desired period the learning phase ceases. However, when longer training is required there is no update in the initial values or not even an informative message expressing the non convergence (or overfitting) for a specific model configuration. It is from the plots of loss and evaluation metrics that one sees whether the model converges or not. An efficient automated

model should (optionally) be able to modify (at leas some) initial hyperparameter values (if such operation is permitted by the user) in order to automatically improve the tuning. Similarly for the values of sequence lengths expressing the length of past (future) information to learn from (inference to). Different values should automatically be tried until performant values can be established.

- Another important element to be taken into consideration concerns the order of the various operations.The currently developed models respect the provided order and sequentially execute the related code. However, a competent model should be able to define this order (progressively) as varying the order of different procedures may interfere to the resulting outputs. For instance, in a model combining Maxpooling and LayerNormalization the output is not necessarily the same in the two cases when *Maxpooling follows LayerNormalizaton* and *LayerNormalizaton follows Maxpooling*. A "smart model" should be able to examine such cases automatically.
- Additionally, it would be very interesting to evaluate a model involving *model ensembling*. Associating the outcomes of different prediction approaches (e.g. taking a weighted average) could lead to performant results. Defining an algorithm estimating the "weights" or even used existing ones could increase the model efficiency ([33]).

3. With the proposed work two distinct approaches are proposed. The first one is based on the perspective that all feature values may influence the learning phase. Thus, all possible values of feature "ORIGIN" are simultaneously utilized by a given model. Within the second approach, a distinct study is considered for each value of feature "ORIGIN", that is for each possible airport from which a flight departs, (entries independence among different values of feature "ORIGIN" is here assumed). From this aspect, more alternatives can be exploited. Instead of "ORIGIN" one could consider the values of feature "DESTINATION", or pairs of values ("ORIGIN", "DESTINATION").

   However, each proposed DNN model can be utilized for any considered approach independently of the involved values of features.

These particular tasks were the principal goals of this study. However, as designing and developing the proposed tool required a lot of work not enough time remained for the conception of a fully automated DNN model. Furthermore, the the potential of the presented models is not appropriately explored as very short training durations are made.

Acknowledgements

Frequently, I was encouraged by Professor Pravin Varaiya, to whom I will remain *eternally thankful*, to learn new performant approaches with the aim of combining stochastic optimal control theory with computation for achieving performant engineering to complex dynamical problems.

Wishing to associate control theory with artificial intelligence, I decided to follow the courses on Machine Learning & Artificial Intelligence available by UC Berkeley. I am deeply grateful to all the persons who organized and taught this exceptional program (gifted UCB Professors, Gabriel Gomes, Josh Hug, Jonathan Kolstad, Reed Walker and the Emeritus Learning Team) allowing a profound exploration and clear understanding of various artificial intelligence approaches.

I would like to express my greatest gratitude to Professors Isaac Arzi and Matilde d'Amelio as well to my learning facilitator Dr Jessica Cervi. Through their great generosity to share precious knowledge with all the students, perform noteworthy, creative recommendations and effective explanations they provided a deep and balanced comprehension among *theory, practical applications and inherent intuition*.

I also wish to express my gratefulness to an aviation expert, captain Gregory Livesay, a man of exception, for his precious clarifications and intelligent advises on flights characteristics.

At my own discretion, I would like to thank my family and friends with a particular consideration to my precious friend Daniel. Their strong support reinforced me to work at my best for this program despite the hard conditions and ordeals I was repeatedly facing through out the year. Finally, a specific thought for Mr. David Rigaudie and his wonderful team (Abbès, Alain, Christian, Cédric, Claudine, Clémence, Fabry, Isabelle, Lyonel, Malaguie, Nelly, Nicolas, Rudy, Sedo, Soad). Their continuous encouragement and assistance helped me maintaining a strong will to successfully complete this course and thus been able to search new professional opportunities in accordance with my dearest hopes.

Please, accept my warmest appreciation and my sincerest promise to prove that all your assistance and patience with me worth the pain.

Thank you very much.

Appendices

A Folders With Experimentations

The folder comprised of the final version for the required files to run a model is going to be described.

Furthermore, as some of the introduced experiments use previous versions of the employed implementations and involve plots and results which weren't discussed in this document the folders with these ressources are also presented. It would be appropriate to rerun all these experiments for the last version of the proposed tool but this was difficult due to time contraints.

A.1 Folder Software

The principal version of the proposed tool to be employed for any implementation is included in folder named "Software". Within this folder the following folders of files are included:

- file "fi_creat_df". This file proceeds to the data treatment and creates a serialized dataframe using Python "pickle" object. This dataframe will then be employed by any proposed approach (single study for all values of feature "ORIGIN" or distinct studies per distinct value of feature "ORIGIN") and models.
- "cc_fi_models_1". This file contains the proposed models defining network topologies.
- "cc_fi_input_variables". This is the file with the variables to initialize before any implementation.
- "cc_fi_fcts_1". The file to execute in order to implement and evaluate any desired model and approach.
- "data_new_1". The folder with the available data downloaded from Kaggle, see [34].

Remark A.1.1

All folders with the experiments employ the same files. It is the involved code which presents differences. Consequently, these common files will not be further referred.

A.2 Folder Code_Experiments

This Folder contains the folders with various experiments which be presented in what follows:

1. v_3_m1_k1. Folder employed for the experiment of "Model Dense 1" using BayesianOptimization Tuner, presented in § 7.2. In this folder, it is contained the folder named "Figures_Model" with all the created plots. File "res_m1_k1.pdf" resumes the progress of the implementation and presents the metrics for the best models.

2. f_5_m1_k1_shuffled. Folder employed for the experiment of "Model Dense 1" using BayesianOptimization Tuner, presented in § 7.2 where the batches of sequences for the train, validation and test datasets are shuffled in order to have distant consecutive sequences regarding time.

3. v_3_m1_k3. Folder employed for the experiment of "Model Dense 1" in association with Gridsearch Tuner, developed in § 7.3.

4. f_1_m2_k1. Folder related to the experiment of "Model Dense 2" employing BayesianOptimization Tuner, developed in § 7.4.

5. f_1_m2_k3. Folder related to the experiment of "Model Dense 2" using Gridsearch Tuner, developed in § 7.5.

6. v_3_m3_k1. Folder associated with the experimentation of Model Dense 3 using BayesianOptimization Tuner, developed in § 7.6.

7. v_3_m3_k3. Folder associated with the experimentation of Model Dense 3 using Gridsearch Tuner, developed in § 7.7.

8. v_3_m4_k1. Folder related to the implementation of "Model SimpleConv1D 1" in association with the BayesianOptimization Tuner, developed in § 7.8

9. f_1_m5_k1_exec_1, f_1_m5_k1_exec_2. Folders employed for the experimentation of "Model SimpleConv1D 2" in association with the BayesianOptimization Tuner, developed in § 7.9.

10. v_3_m6_k1. Folder related to the experimentation of model "Model RNN_model_1" using the BayesianOptimization Tuner, developed in § 7.10.

11. f_5_m6_k1_shuffled. Folder related to the experimentation of model "Model RNN_model_1" using the BayesianOptimization Tuner, developed in § 7.10. In this experimentation the train, validation and test sequences are shuffled so as to disturb time order between consecutive samples.

12. f_1_m7_k1. Folder for the implementation of "Model RNN_model_2" using the BayesianOptimization Tuner, developed in § 7.11.

13. f_5_m8_k1. Folder associated with the experimentation of "RNN_model_3" and the BayesianOptimization Tuner, developed in § 7.12.

14. f_5_m9_k1. Folder experimenting "Model RNN_model_4" using the BayesianOptimization Tuner, developed in § 7.13.

15. f_1_m10_k1. Folder experimenting "Model RNN_model_5" using the BayesianOptimization Tuner, developed in § 7.14.

16. f5_distinct_origs_m1_k1. Folder employed for the experiment of "Model Dense 1" using BayesianOptimization Tuner, presented in § 7.2, involving experiments where distinct trainings and inferences occur per each value of feature "ORIGIN".

17. f5_distinct_origs_m2_k1. Folder related to the experiment of "Model Dense 2" employing BayesianOptimization Tuner, developed in § 7.4, employing distinct dataframes for each origin airport.

18. f5_distinct_origs_m3_k1. Folder associated with the experimentation of Model Dense 3 using BayesianOptimization Tuner, developed in § 7.6, involving distinct dataframes for each origin airport.

19. f5_distinct_origs_m6_k1. Folder related to the experimentation of model "Model RNN_model_1" using the BayesianOptimization Tuner, developed in § 7.10, considering a dataframe per origin airport.

A.3 Resources

For the current work the courses taught during the program "Professional Certificate in Machine Learning and Artificial Intelligence", UC Berkeley, are employed.

Additionally, the book "Deep Learning with Python", advised by the Professors of this program, is also considered, see [35].

Contents

References

[1] F. Chollet, “Keras functional api,” https://keras.io/guides/functional_api/, 2020.

[2] T. O’Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, “Keras-tuner,” https://github.com/keras-team/keras-tuner, 2019.

[3] ——, “Kerastuner,” https://keras.io/api/keras_tuner/tuners/bayesian/r, 2019.

[4] ——, “Kerastuner,” https://keras.io/api/keras_tuner/tuners/grid/, 2019.

[5] F. Chollet, “Kerastuner hypermodels,” https://keras.io/api/keras_tuner/hypermodels/, 2023.

[6] T. O’Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, “Dense layer,” https://keras.io/api/layers/core_layers/dense/, 2019.

[7] ——, “Conv1d layer,” https://keras.io/api/layers/convolution_layers/convolution1d/, 2019.

[8] ——, “Recurrent layers,” https://keras.io/api/layers/recurrent_layers/, 2019.

[9] ——, “Dropout layer,” https://keras.io/api/layers/regularization_layers/dropout/, 2019.

[10] Ba *et al.*, “Layernormalization,” https://keras.io/api/layers/normalization_layers/layer_normalization/, 2016.

[11] T. O’Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, “Maxpooling1d layer,” https://keras.io/api/metrics/regression_metrics/, 2019.

[12] ——, “Globalaveragepooling1d layer,” https://keras.io/api/layers/pooling_layers/global_average_pooling1d/, 2019.

[13] ——, “Keras model,” https://www.tensorflow.org/api_docs/python/tf/keras/Model, 2019.

[14] ——, “Optimizers,” https://keras.io/api/optimizers/, 2019.

[15] ——, “Regression losses,” https://keras.io/api/losses/regression_losses/, 2019.

[16] ——, “Regression metrics,” https://keras.io/api/metrics/regression_metrics/, 2019.

[17] ——, “Batchnormalization layer,” https://keras.io/api/layers/normalization_layers/batch_normalization/, 2019.

[18] Cho *et al.*, “tf.keras.layers.bidirectional,” https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional.

[19] O. Zhang, “Leave one out,” https://contrib.scikit-learn.org/category_encoders/leaveoneout.html, 2022.

[20] Keras, “timeseries_dataset_from_array,” https://www.tensorflow.org/api_docs/python/tf/keras/utils/timeseries_dataset_from_array, 2023.

[21] ——, “Hyperparameters class,” https://keras.io/api/keras_tuner/hyperparameters/.

[22] ——, “Callbacks api,” https://keras.io/api/callbacks/.

[23] ——, “Earlystopping,” https://keras.io/api/callbacks/early_stopping/.

[24] ——, “Bayesianoptimization tuner,” https://keras.io/api/keras_tuner/tuners/bayesian/.

[25] ——, “The tuner classes in kerastuner,” https://keras.io/api/keras_tuner/tuners/.

[26] Kaggle, “Airline delay and cancellation data, 2009 to 2018,” https://www.kaggle.com/datasets/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018.

[27] Refundor, “Late aircraft delay,” https://refundor.com/en/news/what-is-a-late-aircraft-delay-compensation, 2023.

[28] Keras, “Layer activation functions,” https://keras.io/api/layers/activations/.

[29] ——, “Optimizers,” https://keras.io/api/optimizers/.

[30] ——, “Losses,” https://keras.io/api/losses/.

[31] ——, “Metrics,” https://keras.io/api/metrics/.

[32] H. Jin, F. Chollet, Q. Song, and X. Hu, “Autokeras: An automl library for deep learning,” *Journal of Machine Learning Research*, vol. 24, no. 6, pp. 1–6, 2023. [Online]. Available: http://jmlr.org/papers/v24/20-1355.html

[33] M. A. Tambiah and Harvard, “Fast weighted average calculation algorithm,” https://orcid.org/0000-0001-6761-3723, 2021.

[34] Kaggle, “Airline delay and cancellation data, 2009 - 2018,” https://www.kaggle.com/datasets/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018.

[35] F. Chollet, *Deep Learning with Python, Second Edition*. Manning, 2022.