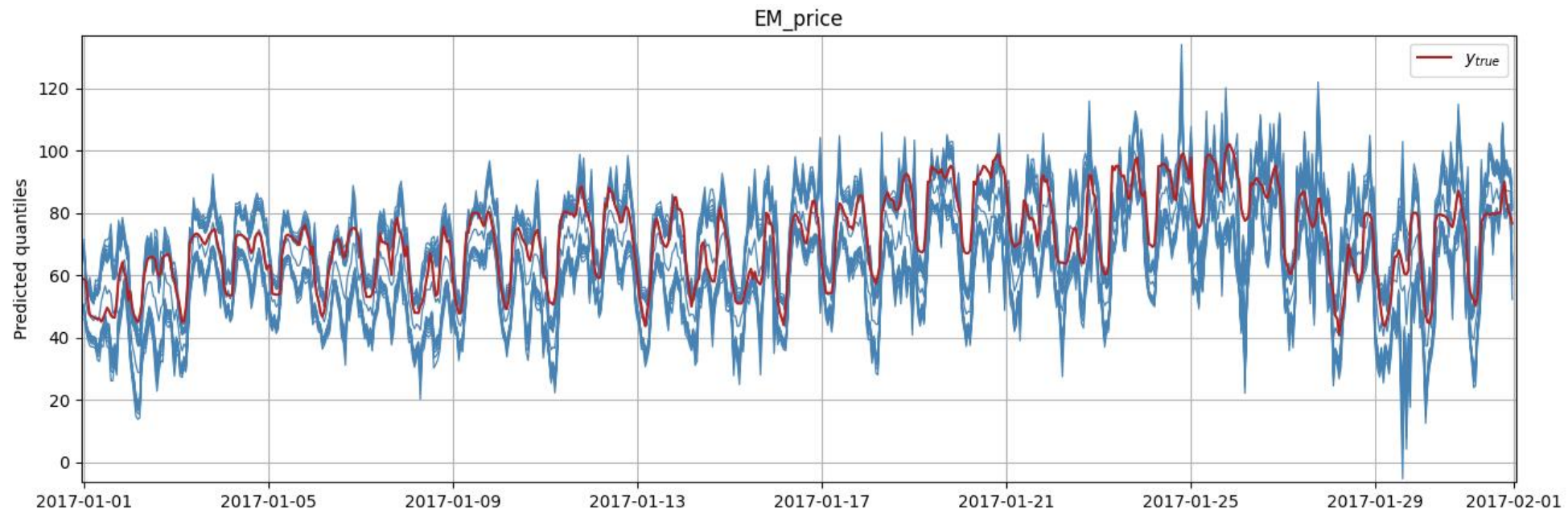


Polimi - Financial Engineering AY 2023/2024

Lab: Electricity Price and Load Forecasting

Goal:

- Learn how to develop probabilistic time series forecasting systems
- Applications to day-ahead electricity price/load forecasting
- Adopting SOTA developments and best practices in the literature (see prelab)
- Focusing ARX-l1(LEAR) and DNN (FFN) models in multi-step settings



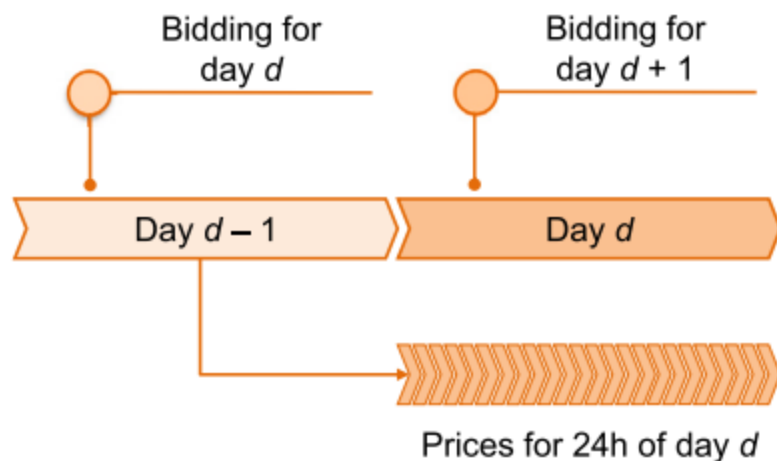
Today:

- Intro to main concepts: recalibration, moving window, batching, etc.
- Overview of the overall codebase (json settings, data processing, etc)
- Step-by-step implementation of the ARX-l1 model in TF and DNN intro

Next (subject to updates):

- L2: hyperparam tuning via optuna
- L3: probabilistic layers and ensembling
- L4: recent conformal inference developments



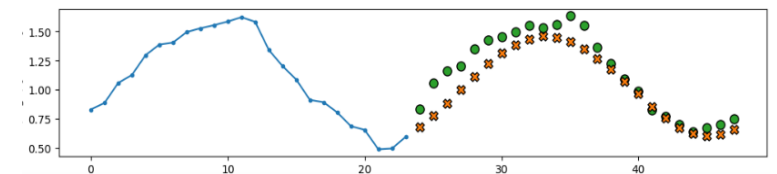
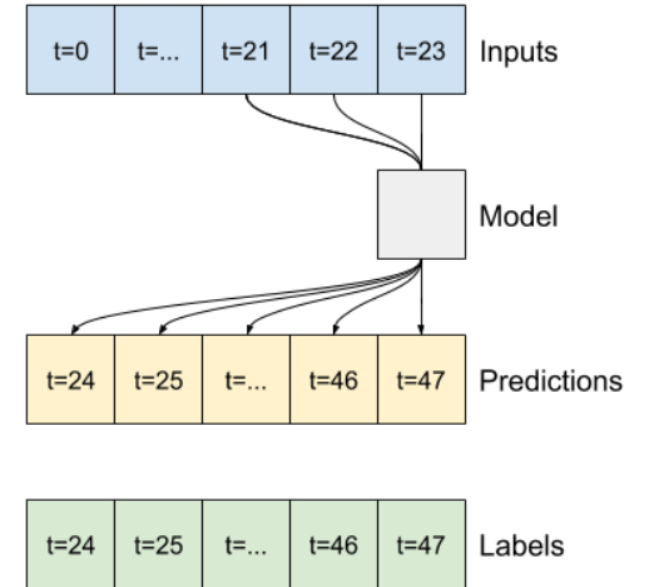
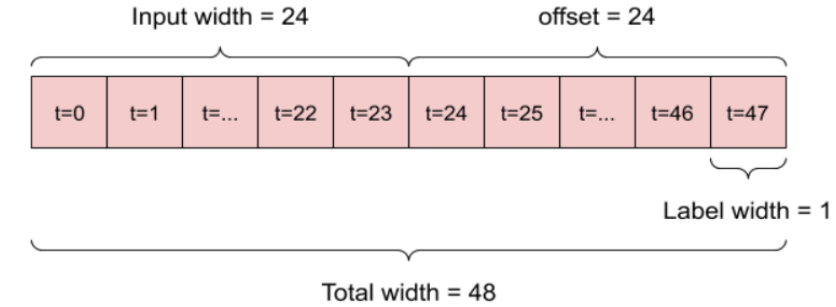


- **Target:** day-ahead hourly prices prediction
- **Input set:** past price values, load forecasts, generation forecasts, etc.
- **Recalibration:** incremental (e.g., daily) model update by including recent information



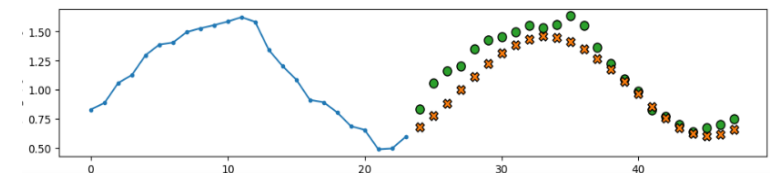
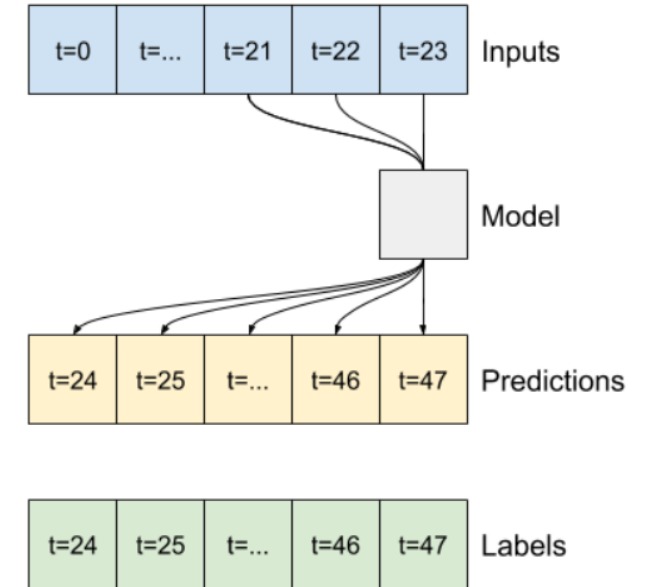
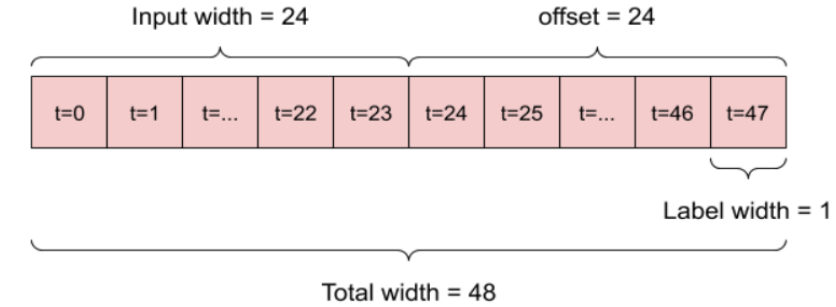
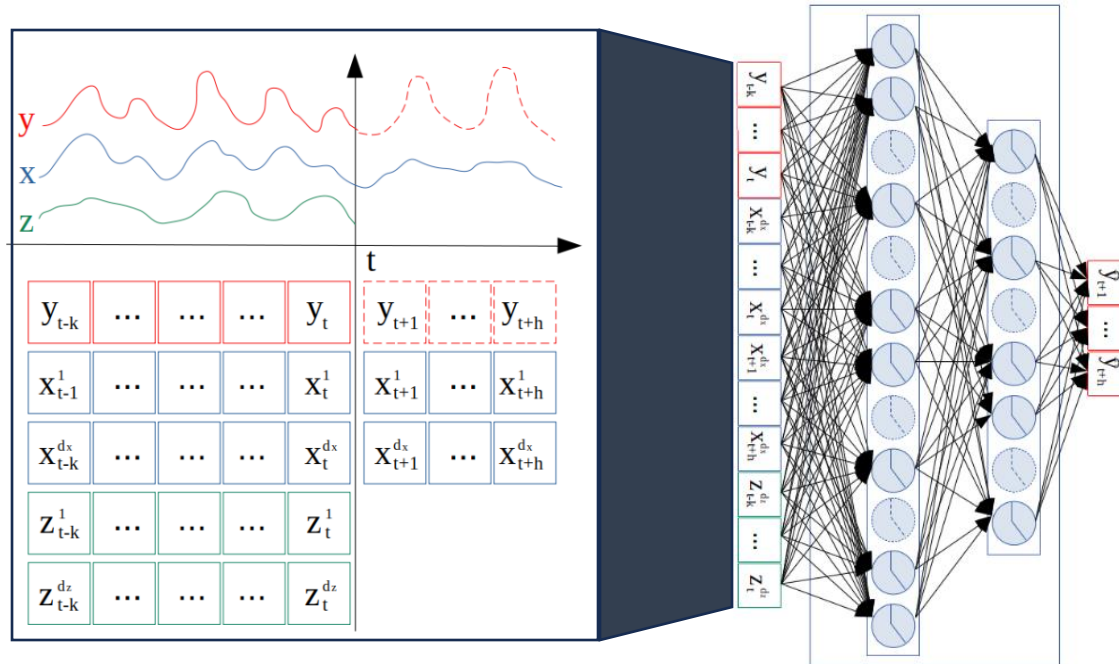
From time series to supervised learning

- **Moving window** to extract batches from TS
- Input **width** and prediction **offset** (multistep)



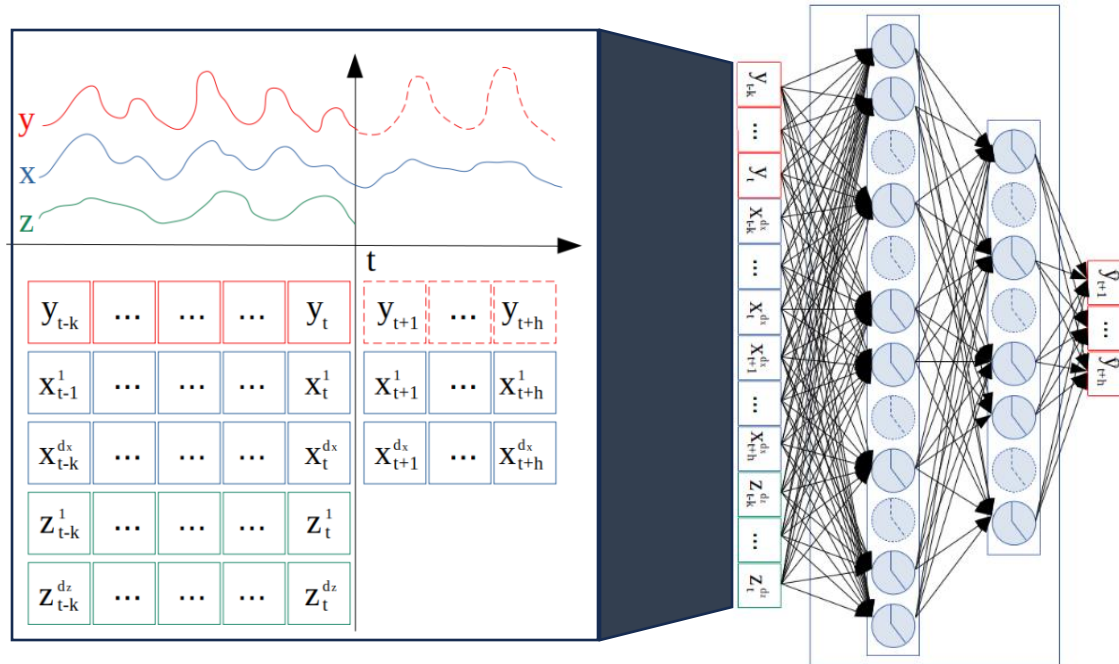
From time series to supervised learning

- Moving window to extract batches from TS
- Input **width** and prediction **offset** (multistep)
- **Past vs future** conditioning sets



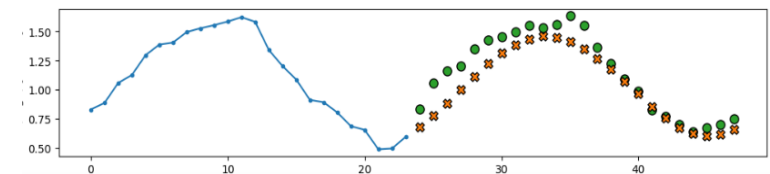
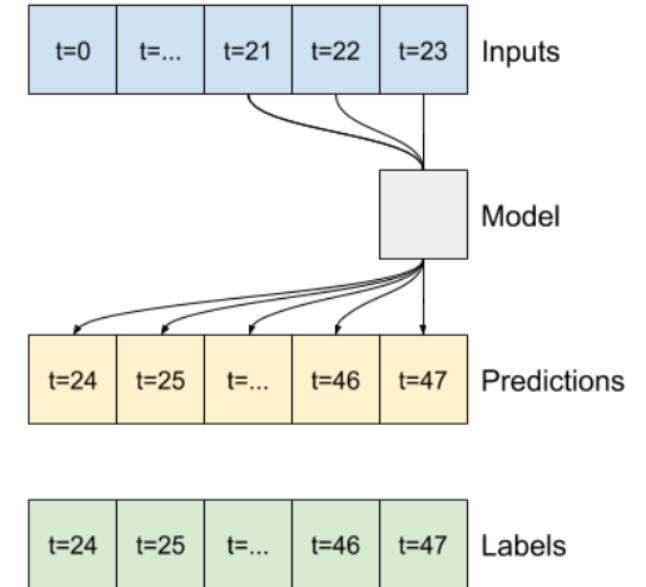
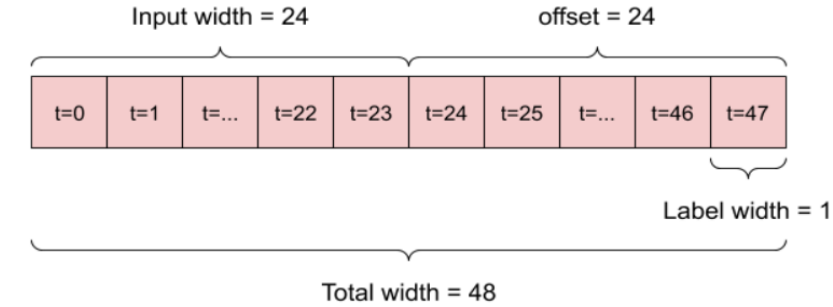
From time series to supervised learning

- **Moving window** to extract batches from TS
- Input **width** and prediction **offset** (multistep)
- **Past vs future** conditioning sets



- Calendar features, e.g., weekday

$$c_t = [\sin(2\pi d_t/6), \cos(2\pi d_t/6)] \quad d_t \in [0, \dots, 6]$$



Codebase overview (goto PyCharm prj)

The image shows a PyCharm IDE window titled "FE24_PEFab_L1 - script_recalibration.py". The interface includes a menu bar (File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help), a toolbar, and a sidebar with "Project", "Structure", and "Bookmarks" views. The "Project" view on the left shows a tree structure for the "FE24_PEFab_L1" project, with a red box highlighting the "tools" directory. The "Structure" view on the right shows the "script_recalibration.py" file. The main editor displays the code for "script_recalibration.py", which includes comments and function calls for loading configurations, instantiating a recalibration engine, and running the recalibration loop.

Project: FE24_PEFab_L1

- data
 - datasets
 - EM_market_2015-01-03_2017-12-31.csv
 - experiments
 - tasks
 - EM_price
 - point-ARX
 - recalib_opt_grid_1_1
 - results
 - exper_configs.json
 - tuned_hyperp-grid_search.json
- tools
 - models
 - ARX.py
 - models_tools.py
 - data_utils.py
 - prediction_quantiles_tools.py
 - PrTSF_Recalib_tools.py
 - LICENSE
 - script_recalibration.py
- External Libraries
- Scratches and Consoles

script_recalibration.py

```
23 plot_weights=True
24 #-----
25
26 # Load experiments configuration from json file
27 configs=load_data_model_configs(task_name=PF_task_name, exper_setup=exper_setup, run_id=run_id)
28
29 # Instantiate recalibration engine
30 PrTSF_eng = PrTsfRecalibEngine(data_configs=configs['data_config'],
31                               model_configs=configs['model_config'])
32
33 # Get model hyperparameters (previously saved or by tuning)
34 model_hyperparams = PrTSF_eng.get_model_hyperparams(method=hyper_mode, optuna_m=configs['model_config']['optuna_m'])
35
36 # Exec recalib loop over the test_set samples, using the tuned hyperparams
37 test_predictions = PrTSF_eng.run_recalibration(model_hyperparams=model_hyperparams,
38                                              plot_history=plot_train_history,
39                                              plot_weights=plot_weights)
40
41 # Plot test predictions
42 plot_quantiles(test_predictions, target=PF_task_name)
43
44 print('Done!')
```


Simple ARX-I1 model in Tensorflow

```
class ARXRegressor:
    def __init__(self, settings, loss):
        self.settings = settings

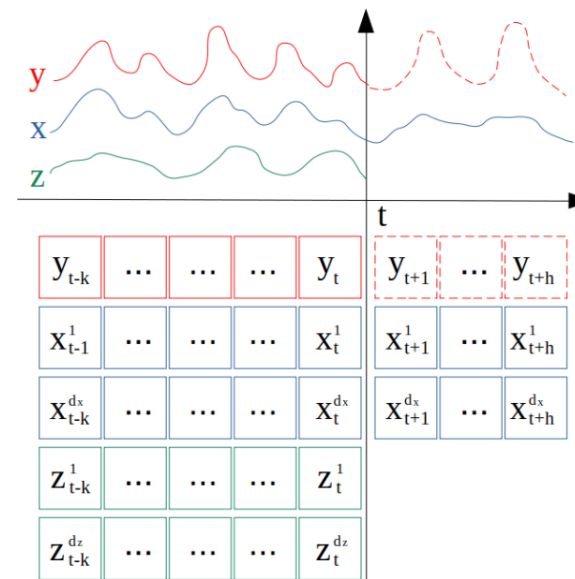
    @staticmethod
    def build_model_input_from_series(x, col_names: List, pred_horiz: int):
        # get index of target and past features
        past_col_idxs = [index for (index, item) in enumerate(col_names)
                        if features_keys['target'] in item or features_keys['past'] in item]

        # get index of const features
        const_col_idxs = [index for (index, item) in enumerate(col_names)
                        if features_keys['const'] in item]

        # get index of futu features
        futu_col_idxs = [index for (index, item) in enumerate(col_names)
                        if features_keys['futu'] in item]

        # build conditioning variables for past features
        past_feat = [x[:, :-pred_horiz, feat_idx] for feat_idx in past_col_idxs]
        # build conditioning variables for futu features
        futu_feat = [x[:, -pred_horiz:, feat_idx] for feat_idx in futu_col_idxs]
        # build conditioning variables for cal features
        c_feat = [x[:, -pred_horiz:-pred_horiz + 1, feat_idx] for feat_idx in const_col_idxs]

        # return flattened input
        return np.concatenate(past_feat + futu_feat + c_feat, axis=1)
```





```
self.__build_model__(loss)

def __build_model__(self, loss):
    x_in = tf.keras.layers.Input(shape=(self.settings['input_size']))
    logit = tf.keras.layers.Dense(self.settings['pred_horiz'],
                                   activation='linear',
                                   kernel_regularizer=tf.keras.regularizers.l1(self.settings['l1']))(x_in)
    output = tf.reshape(logit, (-1, self.settings['pred_horiz'], 1))

    # Create model
    self.model= tf.keras.Model(inputs=[x_in], outputs=[output])

    # Compile the model
    self.model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=self.settings['lr']),
                       loss=loss
    )
```

Simple ARX-I1 model in Tensorflow

```
def fit(self, train_x, train_y, val_x, val_y, verbose=0, pruning_call=None):
    # Convert the data into the input format using the internal converter
    train_x = self.build_model_input_from_series(x=train_x,
                                                col_names=self.settings['x_columns_names'],
                                                pred_horiz=self.settings['pred_horiz'])

    val_x = self.build_model_input_from_series(x=val_x,
                                                col_names=self.settings['x_columns_names'],
                                                pred_horiz=self.settings['pred_horiz'])

    es = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                          patience=self.settings['patience'],
                                          restore_best_weights=False)

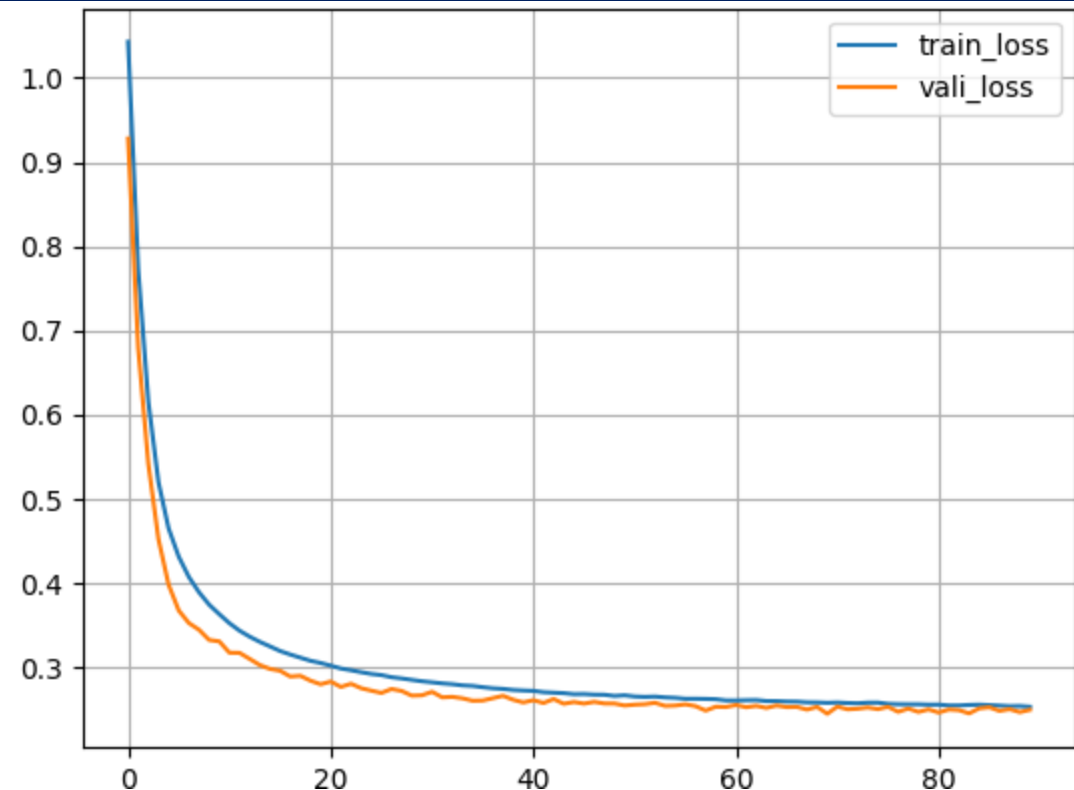
    # Create folder to temporally store checkpoints
    checkpoint_path = os.path.join(os.getcwd(), 'tmp_checkpoints', 'cp.ckpt')
    checkpoint_dir = os.path.dirname(checkpoint_path)
    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    cp = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                          monitor="val_loss", mode="min",
                                          save_best_only=True,
                                          save_weights_only=True, verbose=0)

    if pruning_call==None:
        callbacks = [es, cp]
    else:
        callbacks = [es, cp, pruning_call]

    history = self.model.fit(train_x,
                             train_y,
                             validation_data=(val_x, val_y),
                             epochs=self.settings['max_epochs'],
                             batch_size=self.settings['batch_size'],
                             callbacks=callbacks,
                             verbose=verbose)

    # Load best weights: do not use restore_best_weights from early stop since works only in case it stops training
    self.model.load_weights(checkpoint_path)
    # delete temporary folder
    shutil.rmtree(checkpoint_dir)
    return history
```



2 usages (1 dynamic)

```
def predict(self, x):
    x = self.build_model_input_from_series(x=x,
                                          col_names=self.settings['x_columns_names'],
                                          pred_horiz=self.settings['pred_horiz'])
    return self.model(x)
```

2 usages (1 dynamic)

```
def evaluate(self, x, y):
    x = self.build_model_input_from_series(x=x,
                                          col_names=self.settings['x_columns_names'],
                                          pred_horiz=self.settings['pred_horiz'])
    return self.model.evaluate(x=x, y=y)
```

Run recalibration experiments

```
{
  "data_config": {
    "dataset_name": "EM_market__2015-01-03__2017-12-31.csv",
    "idx_start_train": {
      "y": 2015,
      "m": 1,
      "d": 3
    },
    "idx_start_oos_preds": {
      "y": 2017,
      "m": 1,
      "d": 1
    },
    "idx_end_oos_preds": {
      "y": 2017,
      "m": 1,
      "d": 3
    },
    "keep_past_train_samples": false,
    "steps_lag_win": 2,
    "pred_horiz": 24,
    "preprocess": "StandardScaler",
    "shuffle_mode": "none",
    "num_vali_samples": 100
  },
  "model_config": {
    "PF_method": "point",
    "model_class": "ARX",
    "optuna_m": "grid_search",
    "target_alpha": [
    ],
    "max_epochs": 800,
    "batch_size": 64,
    "patience": 20,
    "num_ense": 1
  }
}
```

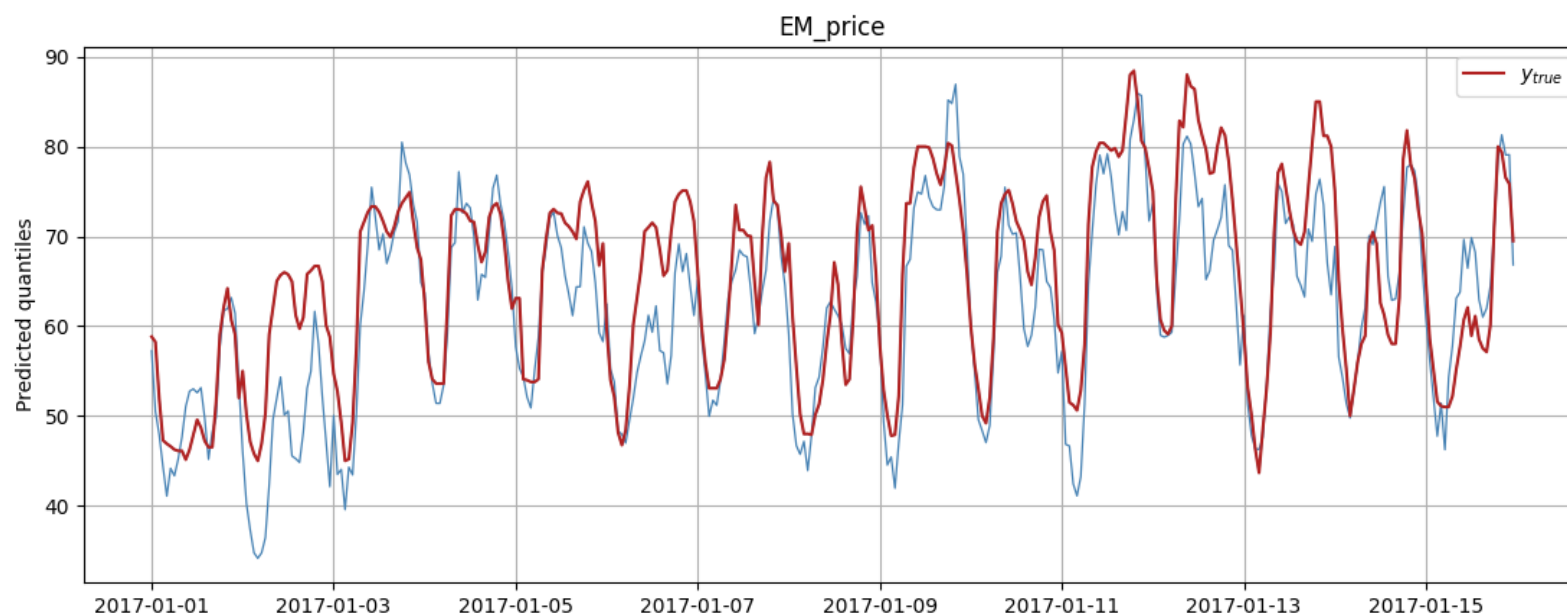
Set experiment
configs in json

Play with the ARX-l1 hyperparams:

- L1 weight
- Learning rate

```
{"l1": 1e-03, "lr": 0.001}
```

Observe the impact on the results



```
logit = tf.keras.layers.Dense(self.settings['pred_horiz'],  
                               activation='linear',  
                               kernel_regularizer=tf.keras.regularizers.l1(self.settings['l1'])  
                               )(x_in)
```

- Experiment ARX-l1 using different l1 values (e.g., 1e-1, 1e-3, 1e-5)
- Plot and analyze the weights (e.g.: get_weights())
- Investigate Ridge/Elastic Net (l1 vs l2 vs l1l2)
- Provide a brief report of the performed experiments and observations

Facultative:

- Implement evaluation metrics (hourly and average): RMSE, MAE, sMAPE
- Compare the impact on the metrics of the hyperparameters setting

Thanks

Alessandro Brusafferri