In [1]:

```
#| default_exp models.informer
```

Informer

The Informer model tackles the vanilla Transformer computational complexity challenges for long-horizon forecasting.

The architecture has three distinctive features:

- A ProbSparse self-attention mechanism with an O time and memory complexity Llog(L).
- A self-attention distilling process that prioritizes attention and efficiently handles long input sequences.
- An MLP multi-step decoder that predicts long time-series sequences in a single forward operation rather than step-by-step.

The Informer model utilizes a three-component approach to define its embedding:

- It employs encoded autoregressive features obtained from a convolution network.
- It uses window-relative positional embeddings derived from harmonic functions.
- Absolute positional embeddings obtained from calendar features are utilized.

References

- Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, Wancai Zhang. "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting"

Figure 1. Temporal Fusion Transformer Architecture.

In [2]:

```
#| export
import math
import numpy as np
from typing import Optional

import torch
import torch.nn as nn

from neuralforecast.common._modules import (
    TransEncoderLayer, TransEncoder,
    TransDecoderLayer, TransDecoder,
    DataEmbedding, AttentionLayer,
)
```

```python
from neuralforecast.common._base_windows import BaseWindows

from neuralforecast.losses.pytorch import MAE
```

In [3]:
```python
#| hide
from fastcore.test import test_eq
from nbdev.showdoc import show_doc
```

1. Auxiliary Functions

In [4]:
```python
#| export
class ConvLayer(nn.Module):
    def __init__(self, c_in):
        super(ConvLayer, self).__init__()
        self.downConv = nn.Conv1d(in_channels=c_in,
                                  out_channels=c_in,
                                  kernel_size=3,
                                  padding=2,
                                  padding_mode='circular')
        self.norm = nn.BatchNorm1d(c_in)
        self.activation = nn.ELU()
        self.maxPool = nn.MaxPool1d(kernel_size=3, stride=2, padding=1)

    def forward(self, x):
        x = self.downConv(x.permute(0, 2, 1))
        x = self.norm(x)
        x = self.activation(x)
        x = self.maxPool(x)
        x = x.transpose(1, 2)
        return x
```

In [5]:
```python
#| export
class ProbMask():
    def __init__(self, B, H, L, index, scores, device="cpu"):
        _mask = torch.ones(L, scores.shape[-1], dtype=torch.bool).to(device)
        _mask_ex = _mask[None, None, :].expand(B, H, L, scores.shape[-1])
        indicator = _mask_ex[torch.arange(B)[:, None, None],
                             torch.arange(H)[None, :, None],
                             index, :].to(device)
        self._mask = indicator.view(scores.shape).to(device)

    @property
    def mask(self):
        return self._mask


class ProbAttention(nn.Module):
    def __init__(self, mask_flag=True, factor=5, scale=None, attention_dropo
        super(ProbAttention, self).__init__()
        self.factor = factor
        self.scale = scale
        self.mask_flag = mask_flag
        self.output_attention = output_attention
        self.dropout = nn.Dropout(attention_dropout)
```

```python
    def _prob_QK(self, Q, K, sample_k, n_top):  # n_top: c*ln(L_q)
        # Q [B, H, L, D]
        B, H, L_K, E = K.shape
        _, _, L_Q, _ = Q.shape

        # calculate the sampled Q_K
        K_expand = K.unsqueeze(-3).expand(B, H, L_Q, L_K, E)

        index_sample = torch.randint(L_K, (L_Q, sample_k))  # real U = U_par
        K_sample = K_expand[:, :, torch.arange(L_Q).unsqueeze(1), index_samp
        Q_K_sample = torch.matmul(Q.unsqueeze(-2), K_sample.transpose(-2, -1

        # find the Top_k query with sparisty measurement
        M = Q_K_sample.max(-1)[0] - torch.div(Q_K_sample.sum(-1), L_K)
        M_top = M.topk(n_top, sorted=False)[1]

        # use the reduced Q to calculate Q_K
        Q_reduce = Q[torch.arange(B)[:, None, None],
                     torch.arange(H)[None, :, None],
                     M_top, :]  # factor*ln(L_q)
        Q_K = torch.matmul(Q_reduce, K.transpose(-2, -1))  # factor*ln(L_q)*

        return Q_K, M_top

    def _get_initial_context(self, V, L_Q):
        B, H, L_V, D = V.shape
        if not self.mask_flag:
            # V_sum = V.sum(dim=-2)
            V_sum = V.mean(dim=-2)
            contex = V_sum.unsqueeze(-2).expand(B, H, L_Q, V_sum.shape[-1]).
        else:  # use mask
            assert (L_Q == L_V)  # requires that L_Q == L_V, i.e. for self-a
            contex = V.cumsum(dim=-2)
        return contex

    def _update_context(self, context_in, V, scores, index, L_Q, attn_mask):
        B, H, L_V, D = V.shape

        if self.mask_flag:
            attn_mask = ProbMask(B, H, L_Q, index, scores, device=V.device)
            scores.masked_fill_(attn_mask.mask, -np.inf)

        attn = torch.softmax(scores, dim=-1)  # nn.Softmax(dim=-1)(scores)

        context_in[torch.arange(B)[:, None, None],
        torch.arange(H)[None, :, None],
        index, :] = torch.matmul(attn, V).type_as(context_in)
        if self.output_attention:
            attns = (torch.ones([B, H, L_V, L_V]) / L_V).type_as(attn).to(at
            attns[torch.arange(B)[:, None, None], torch.arange(H)[None, :, N
            return (context_in, attns)
        else:
            return (context_in, None)

    def forward(self, queries, keys, values, attn_mask):
```

```python
        B, L_Q, H, D = queries.shape
        _, L_K, _, _ = keys.shape

        queries = queries.transpose(2, 1)
        keys = keys.transpose(2, 1)
        values = values.transpose(2, 1)

        U_part = self.factor * np.ceil(np.log(L_K)).astype('int').item()  # 
        u = self.factor * np.ceil(np.log(L_Q)).astype('int').item()  # c*ln(

        U_part = U_part if U_part < L_K else L_K
        u = u if u < L_Q else L_Q

        scores_top, index = self._prob_QK(queries, keys, sample_k=U_part, n_

        # add scale factor
        scale = self.scale or 1. / math.sqrt(D)
        if scale is not None:
            scores_top = scores_top * scale
        # get the context
        context = self._get_initial_context(values, L_Q)
        # update the context with selected top_k queries
        context, attn = self._update_context(context, values, scores_top, in

        return context.contiguous(), attn
```

2. Informer

In [6]:
```python
#| export
class Informer(BaseWindows):
    """ Informer

        The Informer model tackles the vanilla Transformer computational com
        The architecture has three distinctive features:
        1) A ProbSparse self-attention mechanism with an O time and memory c
        2) A self-attention distilling process that prioritizes attention ar
        3) An MLP multi-step decoder that predicts long time-series sequence

    The Informer model utilizes a three-component approach to define its emb
        1) It employs encoded autoregressive features obtained from a convol
        2) It uses window-relative positional embeddings derived from harmon
        3) Absolute positional embeddings obtained from calendar features ar

    *Parameters:*<br>
    `h`: int, forecast horizon.<br>
    `input_size`: int, maximum sequence length for truncated train backpropa
    `futr_exog_list`: str list, future exogenous columns.<br>
    `hist_exog_list`: str list, historic exogenous columns.<br>
    `stat_exog_list`: str list, static exogenous columns.<br>
    `exclude_insample_y`: bool=False, the model skips the autoregressive fea
        `decoder_input_size_multiplier`: float = 0.5, .<br>
    `hidden_size`: int=128, units of embeddings and encoders.<br>
    `n_head`: int=4, controls number of multi-head's attention.<br>
    `dropout`: float (0, 1), dropout throughout Informer architecture.<br>
```

```
        `factor`: int=3, Probsparse attention factor.<br>
        `conv_hidden_size`: int=32, channels of the convolutional encoder.<b
        `activation`: str=`GELU`, activation from ['ReLU', 'Softplus', 'Tanh
    `encoder_layers`: int=2, number of layers for the TCN encoder.<br>
    `decoder_layers`: int=1, number of layers for the MLP decoder.<br>
    `distil`: bool = True, wether the Informer decoder uses bottlenecks.<br>
    `loss`: PyTorch module, instantiated train loss class from [losses colle
    `max_steps`: int=1000, maximum number of training steps.<br>
    `learning_rate`: float=1e-3, Learning rate between (0, 1).<br>
    `num_lr_decays`: int=-1, Number of learning rate decays, evenly distribu
    `early_stop_patience_steps`: int=-1, Number of validation iterations bef
    `val_check_steps`: int=100, Number of training steps between every valic
    `batch_size`: int=32, number of different series in each batch.<br>
    `valid_batch_size`: int=None, number of different series in each validat
    `windows_batch_size`: int=1024, number of windows to sample in each trai
    `inference_windows_batch_size`: int=1024, number of windows to sample ir
    `start_padding_enabled`: bool=False, if True, the model will pad the tim
    `scaler_type`: str='robust', type of scaler for temporal inputs normaliz
    `random_seed`: int=1, random_seed for pytorch initializer and numpy gene
    `num_workers_loader`: int=os.cpu_count(), workers to be used by `TimeSer
    `drop_last_loader`: bool=False, if True `TimeSeriesDataLoader` drops las
    `alias`: str, optional,  Custom name of the model.<br>
    `**trainer_kwargs`: int,  keyword trainer arguments inherited from [PyTo

        *References*<br>
        - [Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li,
    """
    # Class attributes
    SAMPLING_TYPE = 'windows'

    def __init__(self,
                 h: int,
                 input_size: int,
                 stat_exog_list = None,
                 hist_exog_list = None,
                 futr_exog_list = None,
                 exclude_insample_y = False,
                 decoder_input_size_multiplier: float = 0.5,
                 hidden_size: int = 128,
                 dropout: float = 0.05,
                 factor: int = 3,
                 n_head: int = 4,
                 conv_hidden_size: int = 32,
                 activation: str = 'gelu',
                 encoder_layers: int = 2,
                 decoder_layers: int = 1,
                 distil: bool = True,
                 loss = MAE(),
                 valid_loss = None,
                 max_steps: int = 5000,
                 learning_rate: float = 1e-4,
                 num_lr_decays: int = -1,
                 early_stop_patience_steps: int =-1,
                 val_check_steps: int = 100,
                 batch_size: int = 32,
                 valid_batch_size: Optional[int] = None,
```

```
                 windows_batch_size = 1024,
                 inference_windows_batch_size = 1024,
                 start_padding_enabled = False,
                 step_size: int = 1,
                 scaler_type: str = 'identity',
                 random_seed: int = 1,
                 num_workers_loader: int = 0,
                 drop_last_loader: bool = False,
                 **trainer_kwargs):
        super(Informer, self).__init__(h=h,
                                       input_size=input_size,
                                       hist_exog_list=hist_exog_list,
                                       stat_exog_list=stat_exog_list,
                                       futr_exog_list = futr_exog_list,
                                       exclude_insample_y = exclude_insample
                                       loss=loss,
                                       valid_loss=valid_loss,
                                       max_steps=max_steps,
                                       learning_rate=learning_rate,
                                       num_lr_decays=num_lr_decays,
                                       early_stop_patience_steps=early_stop_
                                       val_check_steps=val_check_steps,
                                       batch_size=batch_size,
                                       valid_batch_size=valid_batch_size,
                                       windows_batch_size=windows_batch_size
                                       inference_windows_batch_size = infere
                                       start_padding_enabled=start_padding_e
                                       step_size=step_size,
                                       scaler_type=scaler_type,
                                       num_workers_loader=num_workers_loader
                                       drop_last_loader=drop_last_loader,
                                       random_seed=random_seed,
                                       **trainer_kwargs)

        # Architecture
        self.futr_input_size = len(self.futr_exog_list)
        self.hist_input_size = len(self.hist_exog_list)
        self.stat_input_size = len(self.stat_exog_list)

        if self.stat_input_size > 0:
            raise Exception('Informer does not support static variables yet'

        if self.hist_input_size > 0:
            raise Exception('Informer does not support historical variables

        self.label_len = int(np.ceil(input_size * decoder_input_size_multipl
        if (self.label_len >= input_size) or (self.label_len <= 0):
            raise Exception(f'Check decoder_input_size_multiplier={decoder_i

        if activation not in ['relu', 'gelu']:
            raise Exception(f'Check activation={activation}')

        self.c_out = self.loss.outputsize_multiplier
        self.output_attention = False
        self.enc_in = 1
        self.dec_in = 1
```

```
        # Embedding
        self.enc_embedding = DataEmbedding(c_in=self.enc_in,
                                           exog_input_size=self.hist_input_s
                                           hidden_size=hidden_size,
                                           pos_embedding=True,
                                           dropout=dropout)
        self.dec_embedding = DataEmbedding(self.dec_in,
                                           exog_input_size=self.hist_input_s
                                           hidden_size=hidden_size,
                                           pos_embedding=True,
                                           dropout=dropout)

        # Encoder
        self.encoder = TransEncoder(
            [
                TransEncoderLayer(
                    AttentionLayer(
                        ProbAttention(False, factor,
                                      attention_dropout=dropout,
                                      output_attention=self.output_attention
                        hidden_size, n_head),
                    hidden_size,
                    conv_hidden_size,
                    dropout=dropout,
                    activation=activation
                ) for l in range(encoder_layers)
            ],
            [
                ConvLayer(
                    hidden_size
                ) for l in range(encoder_layers - 1)
            ] if distil else None,
            norm_layer=torch.nn.LayerNorm(hidden_size)
        )
        # Decoder
        self.decoder = TransDecoder(
            [
                TransDecoderLayer(
                    AttentionLayer(
                        ProbAttention(True, factor, attention_dropout=dropou
                        hidden_size, n_head),
                    AttentionLayer(
                        ProbAttention(False, factor, attention_dropout=dropo
                        hidden_size, n_head),
                    hidden_size,
                    conv_hidden_size,
                    dropout=dropout,
                    activation=activation,
                )
                for l in range(decoder_layers)
            ],
            norm_layer=torch.nn.LayerNorm(hidden_size),
            projection=nn.Linear(hidden_size, self.c_out, bias=True)
        )
```

```
    def forward(self, windows_batch):
        # Parse windows_batch
        insample_y    = windows_batch['insample_y']
        #insample_mask = windows_batch['insample_mask']
        #hist_exog     = windows_batch['hist_exog']
        #stat_exog     = windows_batch['stat_exog']

        futr_exog     = windows_batch['futr_exog']

        insample_y = insample_y.unsqueeze(-1) # [Ws,L,1]

        if self.futr_input_size > 0:
            x_mark_enc = futr_exog[:,:self.input_size,:]
            x_mark_dec = futr_exog[:,-(self.label_len+self.h):,:]
        else:
            x_mark_enc = None
            x_mark_dec = None

        x_dec = torch.zeros(size=(len(insample_y),self.h,1)).to(insample_y.d
        x_dec = torch.cat([insample_y[:,-self.label_len:,:], x_dec], dim=1)

        enc_out = self.enc_embedding(insample_y, x_mark_enc)
        enc_out, _ = self.encoder(enc_out, attn_mask=None) # attns visualiza

        dec_out = self.dec_embedding(x_dec, x_mark_dec)
        dec_out = self.decoder(dec_out, enc_out, x_mask=None,
                               cross_mask=None)

        forecast = self.loss.domain_map(dec_out[:, -self.h:])
        return forecast
```

In [7]:
```
show_doc(Informer)
```

Loading [MathJax]/extensions/Safe.js

Out[7]:

Informer

```
 Informer (h:int, input_size:int, stat_exog_list=None,
           hist_exog_list=None, futr_exog_list=None,
           exclude_insample_y=False,
           decoder_input_size_multiplier:float=0.5, hidden
_size:int=128,
           dropout:float=0.05, factor:int=3, n_head:int=4,
           conv_hidden_size:int=32, activation:str='gelu',
           encoder_layers:int=2, decoder_layers:int=1, dis
til:bool=True,
           loss=MAE(), valid_loss=None, max_steps:int=500
0,
           learning_rate:float=0.0001, num_lr_decays:int=-
1,
           early_stop_patience_steps:int=-1, val_check_ste
ps:int=100,
           batch_size:int=32, valid_batch_size:Optional[in
t]=None,
           windows_batch_size=1024, inference_windows_batc
h_size=1024,
           start_padding_enabled=False, step_size:int=1,
           scaler_type:str='identity', random_seed:int=1,
           num_workers_loader:int=0, drop_last_loader:bool
=False,
           **trainer_kwargs)
```

Informer

```
The Informer model tackles the vanilla Transformer computational com
plexity challenges for long-horizon forecasting.
The architecture has three distinctive features:
1) A ProbSparse self-attention mechanism with an O time and memory c
omplexity Llog(L).
2) A self-attention distilling process that prioritizes attention an
d efficiently handles long input sequences.
3) An MLP multi-step decoder that predicts long time-series sequence
s in a single forward operation rather than step-by-step.
```

The Informer model utilizes a three-component approach to define its embedding: 1) It employs encoded autoregressive features obtained from a convolution network. 2) It uses window-relative positional embeddings derived from harmonic functions. 3) Absolute positional embeddings obtained from calendar features are utilized.

Parameters:
`h` : int, forecast horizon.
`input_size` : int, maximum sequence length for truncated train backpropagation. Default -1 uses all history.
`futr_exog_list` : str list, future exogenous columns.
`hist_exog_list` : str list, historic exogenous columns.
`stat_exog_list` : str list, static exogenous columns.
`exclude_insample_y` : bool=False, the model skips the autoregressive features y[t-input_size:t] if True.
`decoder_input_size_multiplier` : float = 0.5, .
`hidden_size` : int=128, units of embeddings and encoders.
`n_head` : int=4, controls number of multi-head's attention.
`dropout` : float (0, 1), dropout throughout Informer architecture.
`factor` : int=3, Probsparse attention factor.
`conv_hidden_size` : int=32, channels of the convolutional encoder.
`activation` : str=`GELU`, activation from ['ReLU', 'Softplus', 'Tanh', 'SELU', 'LeakyReLU', 'PReLU', 'Sigmoid', 'GELU'].
`encoder_layers` : int=2, number of layers for the TCN encoder.
`decoder_layers` : int=1, number of layers for the MLP decoder.
`distil` : bool = True, wether the Informer decoder uses bottlenecks.
`loss` : PyTorch module, instantiated train loss class from losses collection.
`max_steps` : int=1000, maximum number of training steps.
`learning_rate` : float=1e-3, Learning rate between (0, 1).
`num_lr_decays` : int=-1, Number of learning rate decays, evenly distributed across max_steps.
`early_stop_patience_steps` : int=-1, Number of validation iterations before early stopping.
`val_check_steps` : int=100, Number of training steps between every validation loss check.
`batch_size` : int=32, number of different series in each batch.
`valid_batch_size` : int=None, number of different series in each validation and test batch, if None uses batch_size.
`windows_batch_size` : int=1024, number of windows to sample in each training batch, default uses all.
`inference_windows_batch_size` : int=1024, number of windows to sample in each inference batch.
`start_padding_enabled` : bool=False, if True, the model will pad the time series with zeros at the beginning, by input size.
`scaler_type` : str='robust', type of scaler for temporal inputs normalization see temporal scalers.

`random_seed`: int=1, random_seed for pytorch initializer and numpy generators.
`num_workers_loader`: int=os.cpu_count(), workers to be used by `TimeSeriesDataLoader`.
`drop_last_loader`: bool=False, if True `TimeSeriesDataLoader` drops last non-full batch.
`alias`: str, optional, Custom name of the model.
`**trainer_kwargs`: int, keyword trainer arguments inherited from PyTorch Lighning's trainer.

```
*References*<br>
- [Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, Wancai Zhang. "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting"](https://arxiv.org/abs/2012.07436)<br>
```

In [10]:
```
show_doc(Informer.fit, name='Informer.fit')
```

Out[10]:

Informer.fit

> Informer.fit (dataset, val_size=0, test_size=0, random_seed=None)

Fit.

The `fit` method, optimizes the neural network's weights using the initialization parameters (`learning_rate`, `windows_batch_size`, ...) and the `loss` function as defined during the initialization. Within `fit` we use a PyTorch Lightning `Trainer` that inherits the initialization's `self.trainer_kwargs`, to customize its inputs, see PL's trainer arguments.

The method is designed to be compatible with SKLearn-like classes and in particular to be compatible with the StatsForecast library.

By default the `model` is not saving training checkpoints to protect disk memory, to get them change `enable_checkpointing=True` in `__init__`.

Parameters:
`dataset`: NeuralForecast's `TimeSeriesDataset`, see documentation.
`val_size`: int, validation size for temporal cross-validation.
`random_seed`: int=None, random_seed for pytorch initializer and numpy generators, overwrites model.**init**'s.
`test_size`: int, test size for temporal cross-validation.

In [8]:
```
show_doc(Informer.predict, name='Informer.predict')
```

Loading [MathJax]/extensions/Safe.js

Out[8]:

Informer.predict

> Informer.predict (dataset, test_size=None, step_size=1, random_seed=None,
> **data_module_kwargs)

Predict.

Neural network prediction with PL's `Trainer` execution of `predict_step`.

Parameters:
`dataset`: NeuralForecast's `TimeSeriesDataset`, see documentation.
`test_size`: int=None, test size for temporal cross-validation.
`step_size`: int=1, Step size between each window.
`random_seed`: int=None, random_seed for pytorch initializer and numpy generators, overwrites model.**init**'s.
`**data_module_kwargs`: PL's TimeSeriesDataModule args, see documentation.

Usage Example

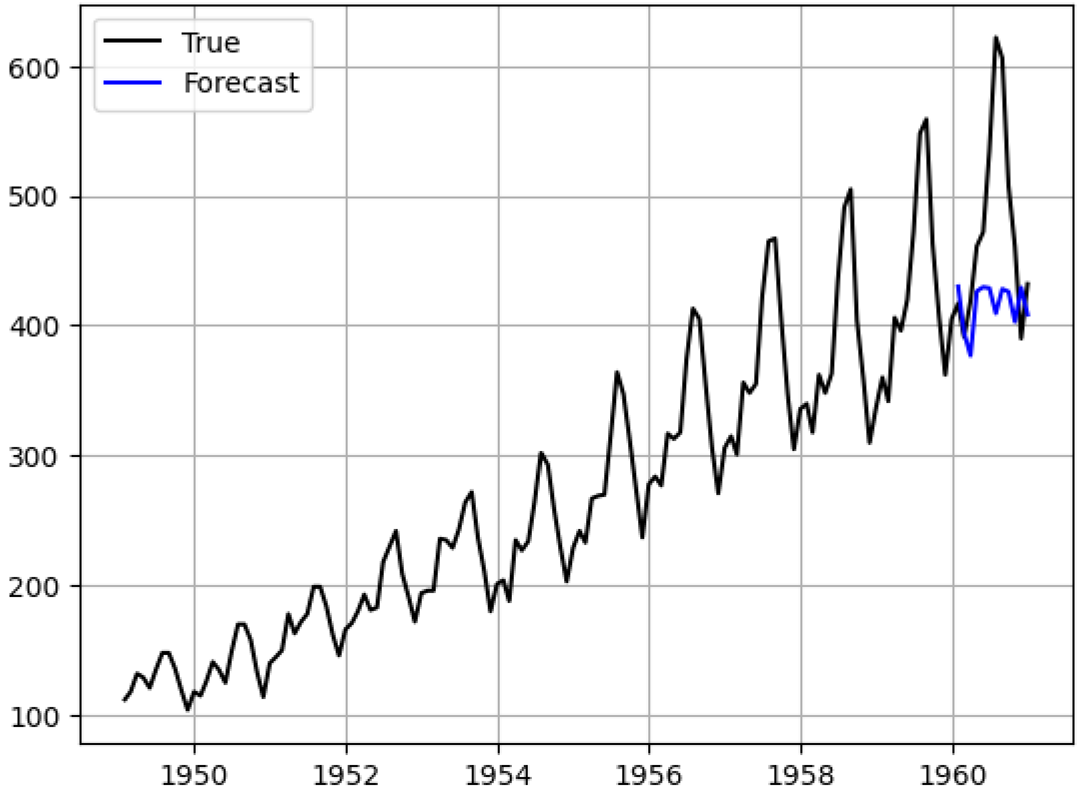
In [9]:

```
#| eval: false
import numpy as np
import pandas as pd
import pytorch_lightning as pl
import matplotlib.pyplot as plt

from neuralforecast import NeuralForecast
from neuralforecast.models import MLP
from neuralforecast.losses.pytorch import MQLoss, DistributionLoss
from neuralforecast.tsdataset import TimeSeriesDataset
from neuralforecast.utils import AirPassengers, AirPassengersPanel, AirPasse

AirPassengersPanel, calendar_cols = augment_calendar_df(df=AirPassengersPane

Y_train_df = AirPassengersPanel[AirPassengersPanel.ds<AirPassengersPanel['ds
Y_test_df = AirPassengersPanel[AirPassengersPanel.ds>=AirPassengersPanel['ds

model = Informer(h=12,
                 input_size=24,
                 hidden_size = 16,
                 conv_hidden_size = 32,
                 n_head = 2,
                 #loss=DistributionLoss(distribution='StudentT', level=[80,
                 loss=MAE(),
                 futr_exog_list=calendar_cols,
                 scaler_type='robust',
                 learning_rate=1e-3,
                 max_steps=5,
```

```
                  val_check_steps=50,
                  early_stop_patience_steps=2)

nf = NeuralForecast(
    models=[model],
    freq='M'
)
nf.fit(df=Y_train_df, static_df=AirPassengersStatic, val_size=12)
forecasts = nf.predict(futr_df=Y_test_df)

Y_hat_df = forecasts.reset_index(drop=False).drop(columns=['unique_id','ds']
plot_df = pd.concat([Y_test_df, Y_hat_df], axis=1)
plot_df = pd.concat([Y_train_df, plot_df])

if model.loss.is_distribution_output:
    plot_df = plot_df[plot_df.unique_id=='Airline1'].drop('unique_id', axis=
    plt.plot(plot_df['ds'], plot_df['y'], c='black', label='True')
    plt.plot(plot_df['ds'], plot_df['Informer-median'], c='blue', label='mec
    plt.fill_between(x=plot_df['ds'][-12:],
                    y1=plot_df['Informer-lo-90'][-12:].values,
                    y2=plot_df['Informer-hi-90'][-12:].values,
                    alpha=0.4, label='level 90')
    plt.grid()
    plt.legend()
    plt.plot()
else:
    plot_df = plot_df[plot_df.unique_id=='Airline1'].drop('unique_id', axis=
    plt.plot(plot_df['ds'], plot_df['y'], c='black', label='True')
    plt.plot(plot_df['ds'], plot_df['Informer'], c='blue', label='Forecast')
    plt.legend()
    plt.grid()
```

```
Seed set to 1
2023-11-02 06:31:43.955641: I tensorflow/core/util/port.cc:111] oneDNN custo
m operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn t
hem off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-02 06:31:43.991615: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could n
ot find cuda drivers on your machine, GPU will not be used.
2023-11-02 06:31:44.161895: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register fa
ctory for plugin cuDNN when one has already been registered
2023-11-02 06:31:44.161926: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_fft.cc:609] Unable to register cuFFT factory: Attempting to register fac
tory for plugin cuFFT when one has already been registered
2023-11-02 06:31:44.163496: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to register
factory for plugin cuBLAS when one has already been registered
2023-11-02 06:31:44.250193: I tensorflow/core/platform/cpu_feature_guard.cc:
182] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operation
s, rebuild TensorFlow with the appropriate compiler flags.
2023-11-02 06:31:45.304798: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Could not find TensorRT
```

```
Sanity Checking: |
| 0/? [00:00…
Training: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Predicting: |
| 0/? [00:00…
```



In [18]:
```
plot_df
```

Loading [MathJax]/extensions/Safe.js

Out[18]:

	ds	y	trend	y_[lag12]	month	Informer
0	1949-01-31	112.0	0	112.0	-0.500000	NaN
1	1949-02-28	118.0	1	118.0	-0.409091	NaN
2	1949-03-31	132.0	2	132.0	-0.318182	NaN
3	1949-04-30	129.0	3	129.0	-0.227273	NaN
4	1949-05-31	121.0	4	121.0	-0.136364	NaN
...	...	...	...	...	...	...
7	1960-08-31	606.0	139	559.0	0.136364	428.100037
8	1960-09-30	508.0	140	463.0	0.227273	426.027283
9	1960-10-31	461.0	141	407.0	0.318182	402.891663
10	1960-11-30	390.0	142	362.0	0.409091	429.251587
11	1960-12-31	432.0	143	405.0	0.500000	408.357788

144 rows × 6 columns

In [27]:

```
y_true = Y_test_df.y.values
y_hat = Y_hat_df['Informer'].values
```

In [28]:

```
from neuralforecast.losses.numpy import mae, mse

print('MAE: ', mae(y_hat, y_true))
print('MSE: ', mse(y_hat, y_true))
```

```
MAE:  69.60045496622722
MSE:  8776.258259076121
```

Informer Implementation

Exchange rate

In [18]:

```
import pandas as pd
from neuralforecast import NeuralForecast

Y_df = pd.read_csv("raw_data/df_Exchange.csv")

Y_df['ds'] = pd.to_datetime(Y_df['ds'])

# For this excercise we are going to take 20% of the DataSet
n_time = len(Y_df.ds.unique())
val_size = int(.1 * n_time)
test_size = int(.2 * n_time)

Y_df.groupby('unique_id').head(2)
```

Out[18]:

	unique_id	ds	y
0	0	1990-01-01	0.606785
1	0	1990-01-02	0.570900
7588	1	1990-01-01	-0.361671
7589	1	1990-01-02	-0.367639
15176	2	1990-01-01	0.735367
15177	2	1990-01-02	0.729629
22764	3	1990-01-01	-1.164373
22765	3	1990-01-02	-1.170907
30352	4	1990-01-01	2.851890
30353	4	1990-01-02	2.851890
37940	5	1990-01-01	-1.861369
37941	5	1990-01-02	-1.838665
45528	6	1990-01-01	-1.820047
45529	6	1990-01-02	-1.847258
53116	OT	1990-01-01	-0.124081
53117	OT	1990-01-02	-0.113588

In [27]:

```
horizon = 96

model = Informer(h=horizon,
                 input_size = horizon,
                 max_steps=100,
                 val_check_steps=10,
                 batch_size = 8,
                 hidden_size = 32,
                 windows_batch_size = 256,
                 early_stop_patience_steps=2)

nf = NeuralForecast(
    models=[model],
    freq='D'
)

Y_hat_df = nf.cross_validation(df=Y_df,
                               val_size=val_size,
                               test_size=test_size,
                               n_windows=None)
```

```
Seed set to 1
Sanity Checking: |
| 0/? [00:00…
Training: |
| 0/? [00:00…
```

```
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Predicting: |
| 0/? [00:00…
```

In [29]:
```
Y_hat_df.to_csv('results/Exchange_rate/Informer.csv')
```

In [28]:
```
from neuralforecast.losses.numpy import mae, mse

print('MAE: ', mae(Y_hat_df['y'], Y_hat_df['Informer']))
print('MSE: ', mse(Y_hat_df['y'], Y_hat_df['Informer']))
```

```
MAE:  0.7146729706209036
MSE:  0.9472710149626793
```

Ettm2

In [16]:
```
import pandas as pd
from neuralforecast.core import NeuralForecast

Y_df = pd.read_csv("raw_data/df_Ettm2.csv")

Y_df['ds'] = pd.to_datetime(Y_df['ds'])

# For this excercise we are going to take 20% of the DataSet
n_time = len(Y_df.ds.unique())
val_size = int(.2 * n_time)
test_size = int(.2 * n_time)

Y_df.groupby('unique_id').head(2)
```

Out[16]:

	unique_id	ds	y
0	HUFL	2016-07-01 00:00:00	-0.041413
1	HUFL	2016-07-01 00:15:00	-0.185467
57600	HULL	2016-07-01 00:00:00	0.040104
57601	HULL	2016-07-01 00:15:00	-0.214450
115200	LUFL	2016-07-01 00:00:00	0.695804
115201	LUFL	2016-07-01 00:15:00	0.434685
172800	LULL	2016-07-01 00:00:00	0.434430
172801	LULL	2016-07-01 00:15:00	0.428168
230400	MUFL	2016-07-01 00:00:00	-0.599211
230401	MUFL	2016-07-01 00:15:00	-0.658068
288000	MULL	2016-07-01 00:00:00	-0.393536
288001	MULL	2016-07-01 00:15:00	-0.659338
345600	OT	2016-07-01 00:00:00	1.018032
345601	OT	2016-07-01 00:15:00	0.980124

In [17]:

```
horizon = 96

model = Informer(h=horizon,
                 input_size = horizon,
                 max_steps=100,
                 val_check_steps=10,
                 early_stop_patience_steps=3)

nf = NeuralForecast(
    models=[model],
    freq='15min'
)

Y_hat_df = nf.cross_validation(df=Y_df,
                               val_size=val_size,
                               test_size=test_size,
                               n_windows=None)
```

```
Seed set to 1
2023-11-02 06:04:25.014417: I tensorflow/core/util/port.cc:111] oneDNN custo
m operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn t
hem off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-02 06:04:25.017559: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could n
ot find cuda drivers on your machine, GPU will not be used.
2023-11-02 06:04:25.052813: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register fa
ctory for plugin cuDNN when one has already been registered
2023-11-02 06:04:25.052850: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_fft.cc:609] Unable to register cuFFT factory: Attempting to register fac
tory for plugin cuFFT when one has already been registered
2023-11-02 06:04:25.052871: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to register
factory for plugin cuBLAS when one has already been registered
2023-11-02 06:04:25.062723: I tensorflow/core/platform/cpu_feature_guard.cc:
182] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operation
s, rebuild TensorFlow with the appropriate compiler flags.
2023-11-02 06:04:26.791222: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Could not find TensorRT
Sanity Checking: |
| 0/? [00:00…
Training: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Predicting: |
| 0/? [00:00…
```

In [11]:
```
Y_hat_df.to_csv('results/Ettm2/Informer.csv')
```

Weather

In [8]:
```
import pandas as pd
from neuralforecast.core import NeuralForecast

Y_df = pd.read_csv("raw_data/df_Weather.csv")

Y_df['ds'] = pd.to_datetime(Y_df['ds'])

# For this excercise we are going to take 20% of the DataSet
n_time = len(Y_df.ds.unique())
val_size = int(.1 * n_time)
test_size = int(.2 * n_time)

Y_df.groupby('unique_id').head(2)
```

Out[8]:

	unique_id	ds	y
0	H2OC (mmol/mol)	2020-01-01 00:10:00	-0.999107
1	H2OC (mmol/mol)	2020-01-01 00:20:00	-1.008072
52695	OT	2020-01-01 00:10:00	0.044395
52696	OT	2020-01-01 00:20:00	0.044134
105390	PAR (�mol/m�/s)	2020-01-01 00:10:00	-0.679493
105391	PAR (�mol/m�/s)	2020-01-01 00:20:00	-0.679493
158085	SWDR (W/m�)	2020-01-01 00:10:00	-0.672767
158086	SWDR (W/m�)	2020-01-01 00:20:00	-0.672767
210780	T (degC)	2020-01-01 00:10:00	-1.459980
210781	T (degC)	2020-01-01 00:20:00	-1.454798
263475	Tdew (degC)	2020-01-01 00:10:00	-1.052596
263476	Tdew (degC)	2020-01-01 00:20:00	-1.069612
316170	Tlog (degC)	2020-01-01 00:10:00	-1.424132
316171	Tlog (degC)	2020-01-01 00:20:00	-1.416612
368865	Tpot (K)	2020-01-01 00:10:00	-1.607935
368866	Tpot (K)	2020-01-01 00:20:00	-1.602882
421560	VPact (mbar)	2020-01-01 00:10:00	-0.979132
421561	VPact (mbar)	2020-01-01 00:20:00	-0.990506
474255	VPdef (mbar)	2020-01-01 00:10:00	-0.838497
474256	VPdef (mbar)	2020-01-01 00:20:00	-0.828332
526950	VPmax (mbar)	2020-01-01 00:10:00	-1.141181
526951	VPmax (mbar)	2020-01-01 00:20:00	-1.138714
579645	max. PAR (�mol/m�/s)	2020-01-01 00:10:00	-0.588296
579646	max. PAR (�mol/m�/s)	2020-01-01 00:20:00	-0.588296
632340	max. wv (m/s)	2020-01-01 00:10:00	-0.832381
632341	max. wv (m/s)	2020-01-01 00:20:00	-1.125140
685035	p (mbar)	2020-01-01 00:10:00	2.114257
685036	p (mbar)	2020-01-01 00:20:00	2.099194
737730	rain (mm)	2020-01-01 00:10:00	-0.093506
737731	rain (mm)	2020-01-01 00:20:00	-0.093506
790425	raining (s)	2020-01-01 00:10:00	-0.221050
790426	raining (s)	2020-01-01 00:20:00	-0.221050
843120	rh (%)	2020-01-01 00:10:00	0.990128

	unique_id	ds	y
843121	rh (%)	2020-01-01 00:20:00	0.942141
895815	rho (g/m**3)	2020-01-01 00:10:00	1.940406
895816	rho (g/m**3)	2020-01-01 00:20:00	1.932788
948510	sh (g/kg)	2020-01-01 00:10:00	-0.998513
948511	sh (g/kg)	2020-01-01 00:20:00	-1.009228
1001205	wd (deg)	2020-01-01 00:10:00	0.555571
1001206	wd (deg)	2020-01-01 00:20:00	0.354339
1053900	wv (m/s)	2020-01-01 00:10:00	-0.017801
1053901	wv (m/s)	2020-01-01 00:20:00	-0.029125

In [9]:

```
horizon = 96

model = Informer(h=horizon,
                input_size = horizon,
                max_steps=100,
                val_check_steps=10,
                 batch_size = 21,
                 hidden_size = 32,
                 windows_batch_size = 256,
                early_stop_patience_steps=2)

nf = NeuralForecast(
    models=[model],
    freq='10min'
)

Y_hat_df = nf.cross_validation(df=Y_df,
                               val_size=val_size,
                               test_size=test_size,
                               n_windows=None)
```

```
Seed set to 1
2023-11-02 16:56:35.610357: I tensorflow/core/util/port.cc:111] oneDNN custo
m operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn t
hem off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-02 16:56:35.629810: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could n
ot find cuda drivers on your machine, GPU will not be used.
2023-11-02 16:56:35.776506: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register fa
ctory for plugin cuDNN when one has already been registered
2023-11-02 16:56:35.776610: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_fft.cc:609] Unable to register cuFFT factory: Attempting to register fac
tory for plugin cuFFT when one has already been registered
2023-11-02 16:56:35.776903: E tensorflow/compiler/xla/stream_executor/cuda/c
uda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to register
factory for plugin cuBLAS when one has already been registered
2023-11-02 16:56:35.840966: I tensorflow/core/platform/cpu_feature_guard.cc:
182] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operation
s, rebuild TensorFlow with the appropriate compiler flags.
2023-11-02 16:56:37.119305: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Could not find TensorRT
Sanity Checking: |
| 0/? [00:00…
Training: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Validation: |
| 0/? [00:00…
Predicting: |
| 0/? [00:00…
```

In [10]:

```
from neuralforecast.losses.numpy import mae, mse

print('MAE: ', mae(Y_hat_df['y'], Y_hat_df['Informer']))
print('MSE: ', mse(Y_hat_df['y'], Y_hat_df['Informer']))
```

```
MAE:  0.3196947730662484
MSE:  0.2577551046103829
```

In [11]:
```
Y_hat_df.to_csv('results/Weather/Informer.csv')
```

In [13]:
```
data = {'Informer_MSE': mse(Y_hat_df['y'], Y_hat_df['Informer']),
        'Informer_MAE': mae(Y_hat_df['y'], Y_hat_df['Informer'])}

df = pd.DataFrame(data, index=['Weather'])
df.to_csv('results/Weather/df_Informer.csv')
```

In [ ]: