Encoder Decoder Networks for Cambridge UK Weather Time Series

Encoder decoder models for time series analysis of Cambridge UK temperature measurements taken at the University computer lab weather station.

This notebook is being developed on Google Colab, primarily using keras/tensorflow. Initially I was most interested in short term temperature forecasts (less than 2 hours) but now mostly produce results up to 48 hours in the future for comparison with earlier baselines.

See my previous notebooks, web apps etc:

- Cambridge UK temperature forecast python notebooks
- Cambridge UK temperature forecast R models
- Bayesian optimisation of prophet temperature model
- Cambridge University Computer Laboratory weather station R shiny web app

The linked notebooks, web apps etc contain further details including:

- data description
- data cleaning and preparation
- data exploration

In particular, see the notebooks:

- cammet_baselines_2021 including persistent, simple exponential smoothing, Holt Winter's exponential smoothing and vector autoregression
- keras_mlp_fcn_resnet_time_series, which uses a streamlined version of data preparation from Tensorflow time series forecasting tutorial
- lstm_time_series with stacked LSTMs, bidirectional LSTMs and ConvLSTM1D networks
- cnn_time_series with Conv1D, multi-head Conv1D, Conv2D and Inception-style models

Most of the above repositories, notebooks, web apps etc were built on both less data and less thoroughly cleaned data.

Load most of the required packages.

```
import sys
import math
import datetime
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import product
```

```
from sklearn.preprocessing import StandardScaler

import tensorflow as tf

# How to enable Colab GPUs https://colab.research.google.com/notebooks/gpu.ipynb
# Select the Runtime > "Change runtime type" menu to enable a GPU accelerator,
# and then re-execute this cell.
if 'google.colab' in str(get_ipython()):
    device_name = tf.test.gpu_device_name()
    if device_name != '/device:GPU:0':
        raise SystemError('GPU device not found')
    print('Found GPU at: {}'.format(device_name))
    gpu_info = !nvidia-smi
    gpu_info = '\n'.join(gpu_info)
    print(gpu_info)

# try:
#   tpu = tf.distribute.cluster_resolver.TPUClusterResolver()  # TPU detection
#   print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
# except ValueError:
#   raise BaseException('ERROR: Not connected to a TPU runtime; please see the pre
# tf.config.experimental_connect_to_cluster(tpu)
# tf.tpu.experimental.initialize_tpu_system(tpu)
# tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)

import tensorflow.keras as keras
from keras.models import Sequential, Model #, Input
from keras.layers import Input, InputLayer, Layer, Dense, Dropout, Activation, \
                         Flatten, Reshape, LSTM, RepeatVector, Conv1D, \
                         TimeDistributed, Bidirectional, Dropout, \
                         MaxPooling1D, MaxPooling2D, Conv2D, Attention, \
                         Concatenate, Lambda, AdditiveAttention, \
                         GlobalAveragePooling1D, MultiHeadAttention, \
                         LayerNormalization, Embedding
from keras.layers.merge import concatenate  # comment this out for 2.10 to work
from keras.constraints import maxnorm
from keras import regularizers
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Reduces variance in results but won't eliminate it :-(
%env PYTHONHASHSEED=0
import random
random.seed(42)
np.random.seed(42)
tf.random.set_seed(42)

%matplotlib inline
```

```
Found GPU at: /device:GPU:0
Thu Sep 29 08:35:44 2022
+-----------------------------------------------------------------------------
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2
|-------------------------------+----------------------+----------------------
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
```

```
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|                               |                      |               MIG M.
|===============================+======================+======================
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0
| N/A   38C    P0    32W / 250W |    389MiB / 16280MiB |      1%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------

+-----------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|        ID   ID                                                   Usage
|=============================================================================
+-----------------------------------------------------------------------------
env: PYTHONHASHSEED=0
```

Import Data

The measurements are relatively noisy and there are usually several hundred missing values every year; often across multiple variables. Observations have been extensively cleaned but may still have issues. Interpolation and missing value imputation have been used to fill all missing values. See the cleaning section in the Cambridge Temperature Model repository for details. Observations start in August 2008 and end in April 2021 and occur every 30 mins.

```
if 'google.colab' in str(get_ipython()):
    data_loc = "https://github.com/makeyourownmaker/CambridgeTemperatureNotebooks/
else:
    data_loc = "../data/CamMetPrepped2021.04.26.csv"

df = pd.read_csv(data_loc, index_col=['ds'], parse_dates=['ds', 'ds.1'])
df.rename(columns={'ds.1': 'ds'}, inplace = True)
df_orig = df

print("Shape:")
print(df.shape)
print("\nInfo:")
print(df.info())
print("\nSummary stats:")
display(df.describe())
print("\nRaw data:")
display(df)
print("\n")


def plot_examples(data, x_var):
    """Plot 9 sets of observations in 3 * 3 matrix"""

    assert len(data) == 9

    cols = [col for col in data[0].columns if col != x_var]

    fig, axs = plt.subplots(3, 3, figsize = (15, 10))
```

```
    axs = axs.ravel()  # apl for the win :-)

    for i in range(9):
        for col in cols:
            axs[i].plot(data[i][x_var], data[i][col])
            axs[i].xaxis.set_tick_params(rotation = 20, labelsize = 10)

    fig.legend(cols, loc = 'upper center',  ncol = len(cols))

    return None


cols = ['ds', 'y', 'humidity', 'dew.point', # 'pressure',
        'wind.x', 'wind.y', 'day.sin', 'day.cos', 'year.sin', 'year.cos']
ex_plots = 9
hour_window = 24
starts = df.sample(n = ex_plots).index
p_data = [df.loc[starts[i]:starts[i] + datetime.timedelta(hours = hour_window), co
          for i in range(ex_plots)]
plot_examples(p_data, 'ds')
```

```
Shape:
(223250, 13)

Info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 223250 entries, 2008-08-01 00:30:00 to 2021-04-26 01:00:00
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype         
---  ------             --------------   -----         
 0   ds                 223250 non-null  datetime64[ns]
 1   y                  223250 non-null  float64       
 2   humidity           223250 non-null  float64       
 3   dew.point          223250 non-null  float64       
 4   pressure           223250 non-null  float64       
 5   wind.speed.mean    223250 non-null  float64       
 6   wind.bearing.mean  223250 non-null  float64       
 7   wind.x             223250 non-null  float64       
 8   wind.y             223250 non-null  float64       
 9   day.sin            223250 non-null  float64       
 10  day.cos            223250 non-null  float64       
 11  year.sin           223250 non-null  float64       
 12  year.cos           223250 non-null  float64       
dtypes: datetime64[ns](1), float64(12)
memory usage: 23.8 MB
None

Summary stats:
```

	y	humidity	dew.point	pressure	wind.speed.mean	w
count	223250.000000	223250.000000	223250.000000	223250.000000	223250.000000	
mean	10.000512	78.689959	58.880634	1014.336135	4.432390	
std	6.496255	17.274417	51.630120	11.935364	4.013553	
min	-7.000000	20.000000	-100.000000	963.000000	0.000000	
25%	5.200000	68.000000	20.000000	1008.000000	1.200000	
50%	9.600000	83.000000	60.000000	1016.000000	3.500000	
75%	14.500000	92.000000	97.000000	1022.000000	6.600000	
max	36.100000	100.000000	209.000000	1051.000000	29.200000	

```
Raw data:
```

	ds	y	humidity	dew.point	pressure	wind.speed.mean	wind.be
ds							
2008-08-01 00:30:00	2008-08-01 00:30:00	19.5	65.75000	119.150000	1014.416667	1.150000	
2008-08-01 01:00:00	2008-08-01 01:00:00	19.1	49.75000	79.200000	1014.384615	1.461538	
2008-08-01 01:30:00	2008-08-01 01:30:00	19.1	66.17875	106.600000	1014.500000	1.508333	

2008-08-01 02:00:00	2008-08-01 02:00:00	19.1	58.50000	99.250000	1014.076923	1.430769
2008-08-01 02:30:00	2008-08-01 02:30:00	19.1	66.95000	121.883333	1014.416667	1.133333
...	...	...	...	...	...	...
2021-04-25 23:00:00	2021-04-25 23:00:00	3.6	61.00000	-32.000000	1028.000000	1.400000
2021-04-25 23:30:00	2021-04-25 23:30:00	3.6	64.00000	-26.000000	1028.000000	2.600000
2021-04-26 00:00:00	2021-04-26 00:00:00	3.6	58.00000	-39.000000	1028.000000	4.300000
2021-04-26 00:30:00	2021-04-26 00:30:00	3.2	62.00000	-34.000000	1027.000000	5.400000
2021-04-26 01:00:00	2021-04-26 01:00:00	3.2	62.00000	-34.000000	1027.000000	4.200000

223250 rows × 13 columns

Data augmentation with mixup

Wind velocity vectors were clustered around the 45 degree increments. Data augmentation with the mixup method is carried out to counter this clustering.

From the mixup paper: "mixup trains a neural network on convex combinations of pairs of examples and their labels".

Further details on how I apply the standard mixup technique to time series are included in the Window data section of my keras_mlp_fcn_resnet_time_series notebook.

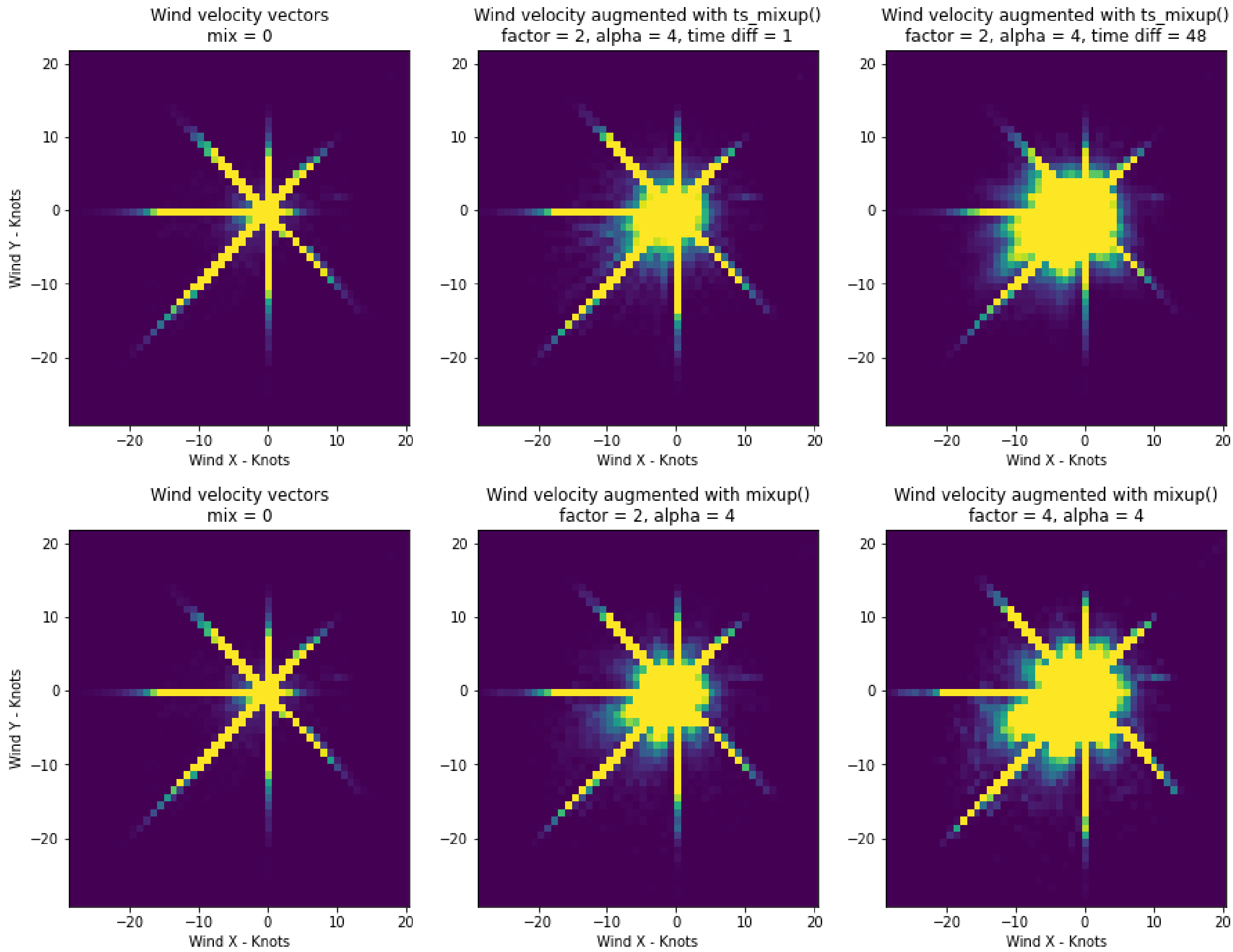
Here is a comparison of the improvement in wind velocity sparsity with standard mixup augmentation and a time series specific mixup.

```
def mixup(data, alpha = 4.0, factor = 1):
    """Augment data with mixup method.

    Standard mixup is applied between randomly chosen observations

    Args:
      data      (pd.DataFrame):    data to run mixup on
      alpha     (float, optional): beta distribution parameter
      factor    (int, optional):   size of mixup dataset to return

    Returns:
      df (pd.DataFrame)

    Notes:
      Duplicates will be removed
      https://arxiv.org/abs/1710.09412
    """
    batch_size = len(data) - 1

    data['epoch'] = data.index.view(np.int64) // 10**9

    # random sample lambda value from beta distribution
    l   = np.random.beta(alpha, alpha, batch_size * factor)
    X_l = l.reshape(batch_size * factor, 1)

    # Get a pair of inputs and outputs
    y1  = data['y'].shift(-1).dropna()
    y1_ = pd.concat([y1] * factor)

    y2  = data['y'][0:batch_size]
    y2_ = pd.concat([y2] * factor)

    X1  = data.drop(columns='y', axis=1).shift(-1).dropna()
    X1_ = pd.concat([X1] * factor)
```

```
    X2  = data.drop(columns='y', axis=1)
    X2  = X2[0:batch_size]
    X2_ = pd.concat([X2] * factor)

    # Perform mixup
    X = X1_ * X_l + X2_ * (1 - X_l)
    y = y1_ * l   + y2_ * (1 - l)

    df = pd.DataFrame(y).join(X)
    df = data.append(df).sort_values('epoch', ascending = True)
    df = df.drop(columns='epoch', axis=1)

    df = df.drop_duplicates(keep = False)

    return df


def ts_mixup(data, alpha = 4.0, factor = 1, time_diff = 1):
    """Augment data with time series mixup method.

    Applies mixup technique to two time series separated by time_diff period.

    Args:
      data      (pd.DataFrame):    data to run mixup on
      alpha     (float, optional): beta distribution parameter
      factor    (int, optional):   size of mixup dataset to return
      time_diff (int, optional):   period between data subsets to run mixup on

    Returns:
      df (pd.DataFrame)

    Notes:
      Duplicates will be removed
      https://arxiv.org/abs/1710.09412
      Standard mixup is applied between randomly chosen observations
    """

    batch_size = len(data) - time_diff

    # Get a pair of inputs and outputs
    y1 = data['y'].shift(-time_diff).dropna()
    y2 = data['y'][0:batch_size]

    X1 = data.drop(columns='y', axis=1).shift(-time_diff).dropna()
    X2 = data.drop(columns='y', axis=1)
    X2 = X2[0:batch_size]

    df = data

    for i in range(factor):
      # random sample lambda value from beta distribution
      l   = np.random.beta(alpha, alpha, 1)
      X_l = np.repeat(l, batch_size).reshape(batch_size, 1)
```

```
        # Perform mixup
        X = X1 * X_l + X2 * (1 - X_l)
        y = y1 * l   + y2 * (1 - l)

        df_new = pd.DataFrame(y).join(X)
        idx_len = np.ceil((df.index[-1] - df.index[0]).days / 365.25)
        df_new.index = df_new.index + pd.offsets.DateOffset(years = idx_len)

        df = df.append(df_new).sort_index(ascending = True)

    df = df.drop_duplicates(keep = False)

    return df


def plot_wind_no_mixup(data, ax):
    """Plot wind vectors without mixup

    Args:
      data      (pd.DataFrame):    wind vector data to plot
      ax        (axes object):     matplotlib axes object for plot
    """

    plt1 = ax.hist2d(data['wind.x'], data['wind.y'], bins = (50, 50), vmax = 400)
    ax.set_xlabel('Wind X - Knots')
    ax.set_ylabel('Wind Y - Knots')
    ax.set_title('Wind velocity vectors\nmix = 0');


def plot_wind_with_mixup(data, ax, mix_func, mix_factor, mix_alpha = 4, mix_td = 1
    """Plot wind vectors with mixup

    Args:
      data       (pd.DataFrame):    wind vector data to plot
      ax         (axes object):     matplotlib axes object for plot
      mix_func   (function)         standard or time series mixup function
      mix_factor (int)              size of mixup dataset to return
      mix_alpha  (int, optional)    beta distribution parameter
      mix_td     (int, optional)    period between data subsets to run mixup on
    """
    title = 'Wind velocity augmented with {0:s}()\n'.format(mix_func)

    if mix_func == 'ts_mixup':
        df_mix = ts_mixup(data.loc[:, ['y', 'wind.x', 'wind.y']],
                          factor = mix_factor,
                          alpha  = mix_alpha,
                          time_diff = mix_td)
        title += 'factor = {0:d}, alpha = {1:d}, time diff = {2:d}'.format(mix_fac
    elif mix_func == 'mixup':
        df_mix = mixup(data.loc[:, ['y', 'wind.x', 'wind.y']],
                       factor = mix_factor,
                       alpha  = mix_alpha)
        title += 'factor = {0:d}, alpha = {1:d}'.format(mix_factor, mix_alpha)
```

```
    plt2 = ax.hist2d(df_mix['wind.x'], df_mix['wind.y'], bins = (50, 50), vmax = 4
    ax.set_xlabel('Wind X - Knots')
    ax.set_title(title);
    # plt.colorbar(plt1, ax = ax3)  # TODO fixme


fig1, (ax11, ax12, ax13) = plt.subplots(1, 3, figsize = (15, 5))
plot_wind_no_mixup(df, ax11)
plot_wind_with_mixup(df, ax12, 'ts_mixup', 2, 4, 1)
plot_wind_with_mixup(df, ax13, 'ts_mixup', 2, 4, 48)

fig2, (ax21, ax22, ax23) = plt.subplots(1, 3, figsize = (15, 5))
plot_wind_no_mixup(df, ax21)
plot_wind_with_mixup(df, ax22, 'mixup', 2)
plot_wind_with_mixup(df, ax23, 'mixup', 4)
```



Mixup improves the categorical legacy of the wind velocity data. Unfortunately, if outliers are present their influence may be reinforced. A priori it's difficult to say which mixup variant is preferable.

Split data¶

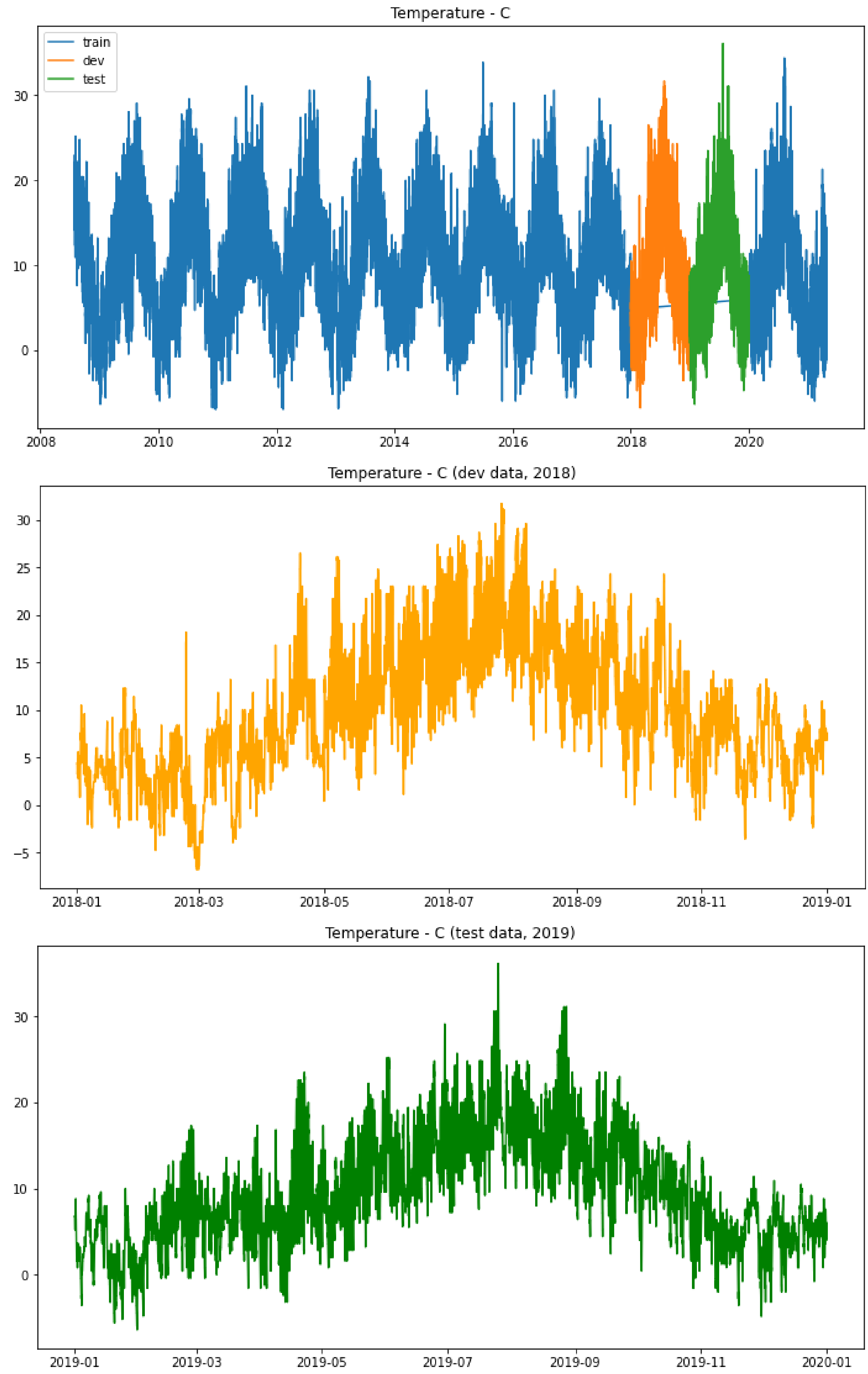
I use data from 2018 for validation, 2019 for testing and the remaining data for training. These are entirely arbitrary choices. This results in an approximate 84%, 8%, 8% split for the training, validation, and test sets respectively.

```
# keep_cols = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y',
#              'day.sin', 'day.cos', 'year.sin', 'year.cos', 'level', 'season1',
#              'season2']

df['year'] = df['ds'].dt.year
train_df = df.loc[(df['year'] != 2018) & (df['year'] != 2019)]
valid_df = df.loc[df['year'] == 2018]
test_df  = df.loc[df['year'] == 2019]
```

```
plt.figure(figsize = (12, 6))
plt.plot(train_df.ds, train_df.y)
plt.plot(valid_df.ds, valid_df.y)
plt.plot(test_df.ds,  test_df.y)
plt.title('Temperature - C')
plt.legend(['train', 'dev', 'test'])
plt.show()

plt.figure(figsize = (12, 6))
plt.plot(valid_df.ds, valid_df.y, color='orange')
plt.title('Temperature - C (dev data, 2018)')
plt.show()

plt.figure(figsize = (12, 6))
plt.plot(test_df.ds, test_df.y, color='green')
plt.title('Temperature - C (test data, 2019)')
plt.show()
```

```
del_cols = ['ds', 'year', 'wind.speed.mean', 'wind.bearing.mean']
train_df = train_df.drop(del_cols, axis = 1)
valid_df = valid_df.drop(del_cols, axis = 1)
test_df  = test_df.drop(del_cols,  axis = 1)
df = df.drop(del_cols, axis = 1)

# ds = {}
models = {}
models['datasets'] = {}
models['datasets']['train'] = train_df
models['datasets']['valid'] = valid_df
```

```
models['datasets']['test']  = test_df

print("df.drop shape: ", df.shape)
print("train shape:   ", train_df.shape)
print("valid shape:   ", valid_df.shape)
print("test shape:    ", test_df.shape)
```


Temperature - C
train
dev
test
Temperature - C (dev data, 2018)
Temperature - C (test data, 2019)

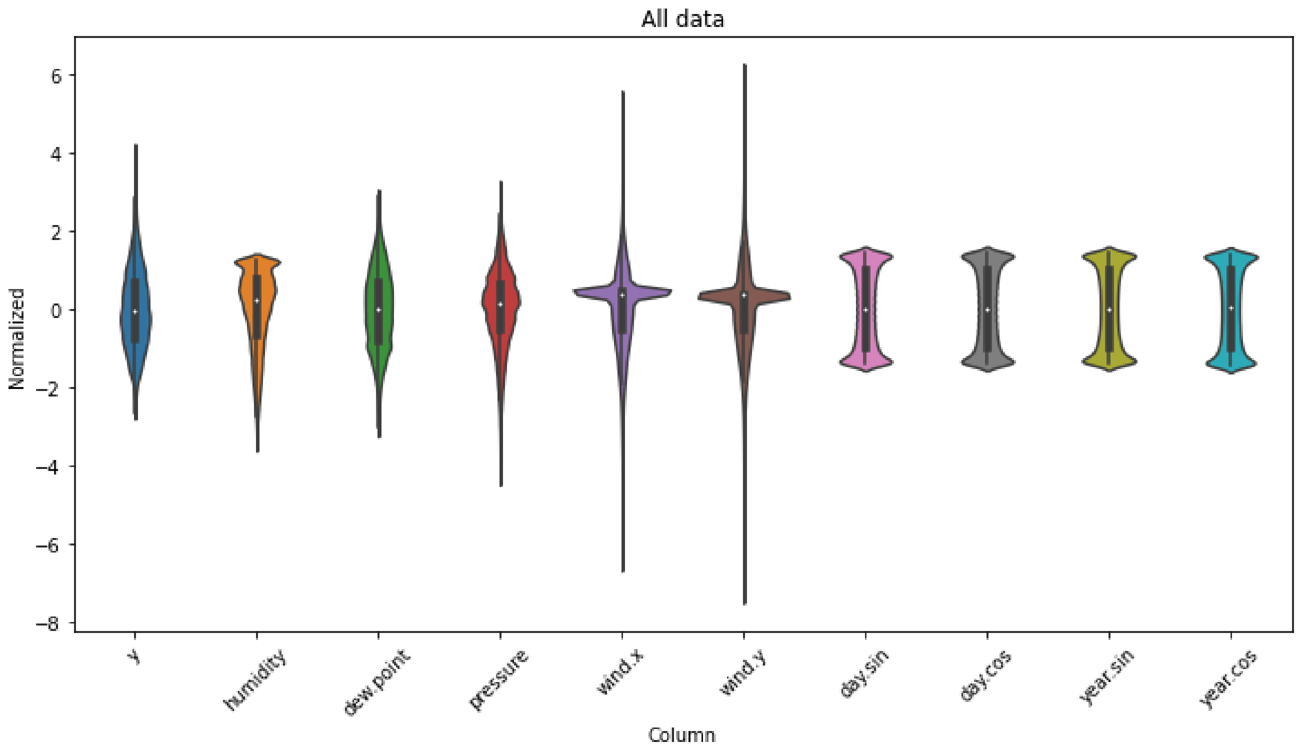
```
df.drop shape:  (223250, 10)
train shape:    (188210, 10)
valid shape:    (17520, 10)
test shape:     (17520, 10)
```

Normalise data

Features should be scaled before neural network training. Arguably, scaling should be done using moving averages to avoid accessing future values. Instead, simple standard score normalisation will be used.

The violin plot shows the distribution of features.

```
def inv_transform(scaler, data, colName, colNames):
    """An inverse scaler for use in model validation section

    For later use in plot_forecasts, plot_horizon_metrics and check_residuals

    See https://stackoverflow.com/a/62170887/100129"""

    dummy = pd.DataFrame(np.zeros((len(data), len(colNames))), columns=colNames)
    dummy[colName] = data
    dummy = pd.DataFrame(scaler.inverse_transform(dummy), columns=colNames)

    return dummy[colName].values


scaler = StandardScaler()
scaler.fit(train_df)

train_df[train_df.columns] = scaler.transform(train_df[train_df.columns] )
valid_df[valid_df.columns] = scaler.transform(valid_df[valid_df.columns] )
test_df[test_df.columns]   = scaler.transform(test_df[test_df.columns] )

df_std = scaler.transform(df)
df_std = pd.DataFrame(df_std)
df_std = df_std.melt(var_name = 'Column', value_name = 'Normalized')

plt.figure(figsize=(12, 6))
ax = sns.violinplot(x = 'Column', y = 'Normalized', data = df_std)
ax.set_xticklabels(df.keys(), rotation = 45)
ax.set_title('All data');
```



Some features have long tails but there are no glaring errors.

Window data

Models are trained using sliding windows of samples from the data.

Window parameters to consider for the [tf.keras.preprocessing.timeseries_dataset_from_array]() function:

- sequence_length:
  - Length of the output sequences (in number of timesteps), or number of **lag** observations to use
- sequence_stride:
  - Period between successive output sequences
  - For stride s, output samples start at index data[i], data[i + s], data[i + 2 * s] etc
  - s can include an **offset** and/or 1 or more **steps ahead** to forecast
- batch_size:
  - Number of samples in each batch
- shuffle:
  - Shuffle output samples, or use chronological order

Initial values used:

- sequence_length (aka lags): 24 (corresponds to 12 hours)

- steps ahead (what to forecast):
    - 48 - 30 mins, 60 mins ... 1,410 mins and 1,440 mins
- offset (space between lags and steps ahead): 0
- batch_size: 16, 32, 64 ...
- shuffle: True for training data

The `make_dataset` function below generates tensorflow datasets for:

- Lags, steps-ahead, offset, batch size and shuffle
- Optionally multiple y columns (Not extensively tested)

Stride is used to specify offset + steps-ahead. Offset will be 0 throughout this notebook.

TODO Insert figure illustrating lags, offsets and steps-ahead.

`shuffle = True` is used with train data. `shuffle = False` is used with validation and test data so the residuals can be checked for heteroscadicity.

Throughout this notebook I use a shorthand notation to describe lags and strides. For example:

- 24l_1s_2m is 24 lags, 1 step ahead, 2 times mixup
- 24l_4s_2m is 24 lags, 4 steps ahead, 2 times mixup

See the `get_model_name` function for details of all abbreviations.

Mixup data augmentation

Data augmentation with mixup: Beyond Empirical Risk Minimization by Zhang *et al* is used to help counter the categorical legacy from the wind bearing variable. Simple 'input mixup' is used as opposed to the batch-based mixup Zhang *et al* focus on. Input mixup has the advantage that it can be used with non-neural network methods. With current settings these datasets are approximately 3 times larger but this can be varied. Three times more training data is manageable on Colab in terms of both training time and memory usage. Test and validation data is left unmodified.

I apply mixup between consecutive observations in the time series instead of the usual random observations. Applying mixup between inputs with equal temperature values will not improve performance and will increase run time.

Here are results for a multi-layer perceptron (MLP) with 24 lags, 1 step ahead, 20 epochs training on both less data and less thoroughly cleaned data.

Augmentation	Train rmse	Train mae	Valid rmse	Valid mae
No augmentation	0.0058	0.053	0.0054	0.052
Input mixup	0.0016	0.025	0.0015	0.025

See this commit for results from other architectures with and without 'input mixup'.

Setup functions for creating windowed datasets.

```
def make_dataset(dataset_params, data):
    assert dataset_params['stride'] >= dataset_params['steps_ahead']
    y_cols = dataset_params['ycols']

    total_window_size = dataset_params['lags'] + dataset_params['stride']

    data = data.drop(columns='epoch', axis = 1, errors = 'ignore')

    if dataset_params['mix_factor'] != 0:
      if dataset_params['mix_type'] == 'ts':
        data_mix = ts_mixup(data,
                            alpha     = dataset_params['mix_alpha'],
                            factor    = dataset_params['mix_factor'],
                            time_diff = dataset_params['mix_diff'])
      else:
        data_mix = mixup(data,
                         alpha  = dataset_params['mix_alpha'],
                         factor = dataset_params['mix_factor'])
    else:
      data_mix = data

    data_mix = data_mix.drop(columns='epoch', axis = 1, errors = 'ignore')
    data_np  = np.array(data_mix, dtype = np.float32)

    ds = tf.keras.preprocessing.timeseries_dataset_from_array(
              data    = data_np,
              targets = None,
              sequence_length = total_window_size,
              sequence_stride = 1,
              shuffle    = dataset_params['shuffle'],
              batch_size = dataset_params['bs'])

    col_indices = {name: i for i, name in enumerate(data.columns)}
    X_slice = slice(0, dataset_params['lags'])
    y_start = total_window_size - dataset_params['steps_ahead']
    y_slice = slice(y_start, None)
    # print(y_start)

    X1_slice = slice(0, dataset_params['lags'])
    # X2_slice = slice(total_window_size, dataset_params['lags'] - 1, -1)  # works
    X2_slice = slice(dataset_params['lags'] - 1, total_window_size - 1)    # new


    def split_window(features):
      X = features[:, X_slice, :]
      y = features[:, y_slice, :]

      # X = tf.stack([X[:, :, col_indices[name]] for name in data.columns],
      #              axis = -1)
      y = tf.stack([y[:, :, col_indices[name]] for name in y_cols],
                  axis = -1)

      # Slicing doesn't preserve static shape info, so set the shapes manually.
      # This way the `tf.data.Datasets` are easier to inspect.
      X.set_shape([None, dataset_params['lags'],        None])
```

```
    y.set_shape([None, dataset_params['steps_ahead'], None])

    return X, y


  def split_window_ed(features):
    X1 = features[:, X1_slice, :]
    X2 = features[:, X2_slice, :]     # runs but overfits
    # X2 = features[:,  y_slice, :]   # experiment - massive overfitting as expe
    y  = features[:,  y_slice, :]

    #X1 = tf.stack([X1[:, :, col_indices[name]] for name in data.columns],
    #              axis = -1)
    y  = tf.stack([y[:, :, col_indices[name]] for name in y_cols],
                 axis = -1)
    X2 = tf.stack([X2[:, :, col_indices[name]] for name in y_cols],
                 axis = -1)

    # Slicing doesn't preserve static shape info, so set the shapes manually.
    # This way the `tf.data.Datasets` are easier to inspect.
    X1.set_shape([None, dataset_params['lags'],        None])
    X2.set_shape([None, dataset_params['steps_ahead'], None])
    y.set_shape( [None, dataset_params['steps_ahead'], None])

    return (X1, X2), y

  if dataset_params['model_type'] in ['encdec', 'transed']:
    ds = ds.map(split_window_ed)
    # Extracting past features + deterministic future + labels
    #ds = ds.map(lambda k: ((k[:-24],
    #                        k[-48:, -1:]),
    #                       k[-48:, 0]))
  else:
    ds = ds.map(split_window)

  return ds


def get_model_name(models, ds_name_params):
  cols = models['datasets']['train'].loc[:, ds_name_params['xcols']].columns

  suffix = "_{0:d}l_{1:d}s".format(ds_name_params['lags'],
                                   ds_name_params['steps_ahead'])

  suffix += "_{0:d}bs".format(ds_name_params['bs'])

  if ds_name_params['feat_maps'] != 0:
    suffix += "_{0:d}fm".format(ds_name_params['feat_maps'])

  if ds_name_params['filters'] != 0:
    suffix += "_{0:d}f".format(ds_name_params['filters'])

  if ds_name_params['kern_size'] != 0 and len(ds_name_params['kern_size']) == 1:
    suffix += "_{0:d}ks".format(ds_name_params['kern_size'])
```

```
    if ds_name_params['kern_size'] != 0 and len(ds_name_params['kern_size']) > 1:
      # suffix += "_{0:d}ks".format(ds_name_params['kern_size'])
      # suffix += '_' + '-'.join(ds_name_params['kern_size']) + 'ks'
      suffix += '_' + '-'.join([str(x) for x in ds_name_params['kern_size']]) + 'k

    if ds_name_params['mix_factor'] > 0:
      suffix += "_{0:d}m".format(ds_name_params['mix_factor'])
      suffix += "_{0:f}a".format(ds_name_params['mix_alpha'])
      if ds_name_params['mix_type'] == 'ts':
        suffix += "_{0:d}td".format(ds_name_params['mix_diff'])
      if ds_name_params['mix_type'] == 'input':
        suffix += '_im'

    if 'level' in  cols and 'season1' in cols and 'season2' in cols:
      suffix += '_tbats'

    if ds_name_params['drop_out'] != 0.0:
      suffix += "_{0:.2E}do".format(ds_name_params['drop_out'])

    if ds_name_params['kern_reg'] != 0.0:
      suffix += "_{0:.2E}kr".format(ds_name_params['kern_reg'])

    if ds_name_params['recu_reg'] != 0.0:
      suffix += "_{0:.2E}rr".format(ds_name_params['recu_reg'])

    if len(ds_name_params['ycols']) > 1:
      suffix += "_{0:d}y".format(len(ds_name_params['ycols']))

    if ds_name_params['ks_feats'] > 0:
      suffix += "_{0:d}ksf".format(ds_name_params['ks_feats'])

    if ds_name_params['ks_time'] > 0:
      suffix += "_{0:d}kst".format(ds_name_params['ks_time'])

    if ds_name_params['model_type'] in ['transed', 'transenc']:
      suffix += "_{0:d}tb".format(ds_name_params['trans_blocks'])
      suffix += "_{0:d}h".format(ds_name_params['num_heads'])
      suffix += "_{0:d}hs".format(ds_name_params['head_size'])

    return ds_name_params['model_type'] + suffix


def make_datasets(models, datasets_params):

    train_data = models['datasets']['train'].loc[:, datasets_params['xcols']]
    valid_data = models['datasets']['valid'].loc[:, datasets_params['xcols']]
    test_data  = models['datasets']['test'].loc[:,  datasets_params['xcols']]

    orig_mix = datasets_params['mix_factor']
    ds_train   = make_dataset(datasets_params, train_data)

    datasets_params['shuffle']    = False
    datasets_params['mix_factor'] = 0
```

```
    ds_valid   = make_dataset(datasets_params, valid_data)

    ds_test   = make_dataset(datasets_params, test_data)
    datasets_params['mix_factor'] = orig_mix

    return [ds_train, ds_valid, ds_test]


def dataset_sanity_checks(data, name):
    print(name, "batches: ", data.cardinality().numpy())
    for batch in data.take(1):
        print("\tX (batch_size, time, features): ", batch[0].shape)
        print("\ty (batch_size, time, features): ", batch[1].shape)
        print("\tX[0][0]: ", batch[0][0])
        print("\ty[0][0]: ", batch[1][0])


def plot_dataset_examples(dataset):
    fig, axs = plt.subplots(3, 3, figsize = (15, 10))
    axs = axs.ravel()
    cols = 0

    for batch in dataset.take(1):
        for i in range(9):
          x = batch[0][i].numpy()
          cols = x.shape[1]
          axs[i].plot(x)

    fig.legend(range(1, cols+1), loc = 'upper center',  ncol = cols+1);


def_cols = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', \
            'day.sin', 'day.cos', 'year.sin', 'year.cos']  # def for default
```

Encoder Decoder Model Building

Encoder decoder networks were first implemented in Sequence to Sequence Learning with Neural Networks and Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. These networks commonly use a LSTM to encode the input sequence to a fixed-length vector representation, which is then decoded by another LSTM into an output sequence. The encoder returns its internal state. The encoder outputs are discarded. This state acts as the "context" for the decoder.

These networks can be used for machine translation, free-from question answering and other sequence to sequence (often abbreviated to seq2seq) problems.

TODO Include basic encoder decoder diagram

The following are a few points I consider when building these encoder decoder models.

Forecast horizons:

- next 24 hours - 48 steps ahead

Metrics:

- mse - mean squared error
    - mse used for loss function to avoid potential problems with infinite values from the square root function
    - rmse - root mean squared error is used for comparison with baselines
    - Huber loss may be worth exploring in the future if outliers remain an issue
- mae - median absolute error
- mape - mean absolute percentage error
    - Not used - mape fails when values, like temperature, become zero

Model enhancements:

- Teacher forcing
    - ...
- Positional encoding
    - ...
- Sequence masking?
    - ...
- mixup?
    - input mixup
    - trialed on final model
    - factor - 2
    - alpha - 4 (recommended in the original publication)
    - time series mixup:
        - time diff - 1, ..., 48
            - period between 2 data subsets to run mixup on

Parameters to consider optimising:

- Learning rate - use LRFinder
- Optimiser - stick with Adam
- Shuffle - true for training
- batch size - 16, 32, 64 ...
- Number of feature maps
    - 8, 16, 32 ...
- epochs

- training shows quite fast convergence so epochs is initially kept quite low (5 or 10)

Model architectures considered:

1. Simplified encoder decoder
    - ...
2. Autoencoder
    - with attention
    - ...
3. Encoder decoder
    - teacher forcing
    - autoregressive inference
    - ...
4. Transformer
    - teacher forcing
    - autoregressive inference
    - MultiHeadAttention
    - positional encoding
    - masking
    - ...
5. Encoder only transformers
    - MultiHeadAttention
    - ...

Learning rate finder

Leslie Smith was one of the first people to work on finding optimal learning rates for deep learning networks in [Cyclical Learning Rates for Training Neural Networks](). Jeremy Howard from [fast.ai]() popularised the learning rate finder used here.

Before building any models, I use a modified version of [Pavel Surmenok's Keras learning rate finder]() to get reasonably close to the optimal learning rate. It's a single small class which I add support for tensorflow datasets to, customise the graphics and add a simple summary function to.

The learning rate finder parameters may benefit from some per-architecture tuning. It's advisable to find a reasonable start_lr value by trying several values which differ by order of magnitude, i.e. 1e-3, 1e-4, 1e-5 etc. It's then worthwhile to use the learning rate finder for fine tuning.

Setup learning rate finder class for later usage:

```python
from keras.callbacks import LambdaCallback
import keras.backend as K


class LRFinder:
    """
    Plots the change of the loss function of a Keras model when the learning rate
    See for details:
    https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neu
    """

    def __init__(self, model):
        self.model  = model
        self.losses = []
        self.lrs    = []
        self.best_lr   = 0.001
        self.best_loss = 1e9


    def on_batch_end(self, batch, logs):
        # Log the learning rate
        lr = K.get_value(self.model.optimizer.lr)
        self.lrs.append(lr)

        # Log the loss
        loss = logs['loss']
        self.losses.append(loss)

        # Check whether the loss got too large or NaN
        if batch > 5 and (math.isnan(loss) or loss > self.best_loss * 4):
            self.model.stop_training = True
            return

        if loss < self.best_loss:
            self.best_loss = loss

        # Increase the learning rate for the next batch
        lr *= self.lr_mult
        K.set_value(self.model.optimizer.lr, lr)


    def find_ds(self, train_ds, start_lr, end_lr, batch_size=64, epochs=1, **kw_f
        # If x_train contains data for multiple inputs, use length of the first in
        # Assumption: the first element in the list is single input; NOT a list of
        # N = x_train[0].shape[0] if isinstance(x_train, list) else x_train.shape[
        N = train_ds.cardinality().numpy()

        # Compute number of batches and LR multiplier
        num_batches = epochs * N / batch_size
        self.lr_mult = (float(end_lr) / float(start_lr)) ** (float(1) / float(num_
        # Save weights into a file
        initial_weights = self.model.get_weights()

        # Remember the original learning rate
        original_lr = K.get_value(self.model.optimizer.lr)
```

```
        # Set the initial learning rate
        K.set_value(self.model.optimizer.lr, start_lr)

        callback = LambdaCallback(on_batch_end=lambda batch, logs: self.on_batch_e

        self.model.fit(train_ds,
                       batch_size=batch_size, epochs=epochs,
                       callbacks=[callback],
                       **kw_fit)

        # Restore the weights to the state before model fitting
        self.model.set_weights(initial_weights)

        # Restore the original learning rate
        K.set_value(self.model.optimizer.lr, original_lr)


    def plot_loss(self, axs, sma, n_skip_beginning, n_skip_end, x_scale='log'):
        """
        Plot the loss.

        Parameters:
            n_skip_beginning - number of batches to skip on the left
            n_skip_end - number of batches to skip on the right
        """
        lrs = self.lrs[n_skip_beginning:-n_skip_end]
        losses = self.losses[n_skip_beginning:-n_skip_end]
        best_lr = self.get_best_lr(sma, n_skip_beginning, n_skip_end)

        axs[0].set_ylabel("loss")
        axs[0].set_xlabel("learning rate (log scale)")
        axs[0].plot(lrs, losses)
        axs[0].vlines(best_lr, np.min(losses), np.max(losses), linestyles='dashed
        axs[0].set_xscale(x_scale)


    def plot_loss_change(self, axs, sma, n_skip_beginning, n_skip_end, y_lim=None
        """
        Plot rate of change of the loss function.

        Parameters:
            axs - subplot axes
            sma - number of batches for simple moving average to smooth out the cu
            n_skip_beginning - number of batches to skip on the left
            n_skip_end - number of batches to skip on the right
            y_lim - limits for the y axis
        """
        derivatives = self.get_derivatives(sma)[n_skip_beginning:-n_skip_end]
        lrs = self.lrs[n_skip_beginning:-n_skip_end]
        best_lr = self.get_best_lr(sma, n_skip_beginning, n_skip_end)
        y_min, y_max = np.min(derivatives), np.max(derivatives)
        x_min, x_max = np.min(lrs), np.max(lrs)
```

```
        axs[1].set_ylabel("rate of loss change")
        axs[1].set_xlabel("learning rate (log scale)")
        axs[1].plot(lrs, derivatives)
        axs[1].vlines(best_lr, y_min, y_max, linestyles='dashed')
        axs[1].hlines(0, x_min, x_max, linestyles='dashed')
        axs[1].set_xscale('log')
        if y_lim == None:
            axs[1].set_ylim([y_min, y_max])
        else:
            axs[1].set_ylim(y_lim)


    def get_derivatives(self, sma):
        assert sma >= 1
        derivatives = [0] * sma
        for i in range(sma, len(self.lrs)):
            derivatives.append((self.losses[i] - self.losses[i - sma]) / sma)

        return derivatives


    def get_best_lr(self, sma, n_skip_beginning, n_skip_end):
        derivatives = self.get_derivatives(sma)
        best_der_idx = np.argmin(derivatives[n_skip_beginning:-n_skip_end])
        # print("sma:", sma)
        # print("n_skip_beginning:", n_skip_beginning)
        # print("n_skip_end:", n_skip_end)
        # print("best_der_idx:", best_der_idx)
        # print("len(derivatives):", len(derivatives))
        # print("derivatives:", derivatives)
        return self.lrs[n_skip_beginning:-n_skip_end][best_der_idx]


    def summarise_lr(self, train_ds, start_lr, end_lr, batch_size=32, epochs=1, sr
        self.find_ds(train_ds, start_lr, end_lr, batch_size, epochs)
        # print("sma:", sma)
        # print("n_skip_beginning:", n_skip_beginning)
        fig, axs = plt.subplots(1, 2, figsize = (9, 6), tight_layout = True)
        axs = axs.ravel()
        self.plot_loss(axs, sma, n_skip_beginning=n_skip_beginning, n_skip_end=5)
        self.plot_loss_change(axs, sma=sma, n_skip_beginning=n_skip_beginning, n_s
        plt.show()

        best_lr = self.get_best_lr(sma=sma, n_skip_beginning=n_skip_beginning, n_s
        print("best lr:", best_lr, "\n")

        self.best_lr = best_lr


def run_lrf(models, params):
    model_name = get_model_name(models, params)

    train_data = models[model_name]['train']
    model = models[model_name]['model']
```

```
    model.compile(loss = 'mse', metrics = ['mae'])
    lrf_inner = LRFinder(model)
    lrf_inner.summarise_lr(train_data, *params['lrf_params'])

    return lrf_inner


lrf_params = [0.000001, 10, 32, 5, 100, 25]
```

Next, define encoder decoder and other network architectures:

- `build_simple_encdec_model`
- `build_encoder_decoder_model`
- `build_autoencoder_model`

```
def get_io_shapes(data, params):
    # print("batches: ", data.cardinality().numpy())

    for batch in data.take(1):
        in_shape  = batch[0][0].shape
        out_shape = batch[1][0].shape

        if params['model_type'] in ['encdec', 'transed']:
          x2_shape = batch[0][1].shape
          return (in_shape[1], in_shape[2]), (x2_shape[1], x2_shape[2]), out_shape

    return in_shape, out_shape


def build_simple_encdec_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data, params)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']

    model = Sequential(name = model_name)
    model.add(InputLayer(input_shape = in_shape))

    # encoder
    model.add(LSTM(feat_maps,
                   activation = 'tanh',
                   input_shape = in_shape))

    # context vector representation
    model.add(RepeatVector(out_steps))

    # decoder
    model.add(LSTM(feat_maps,
                   activation = 'tanh',
                   return_sequences = True))
```

```
    model.add(TimeDistributed(Dense(feat_maps // 2,
                                    activation = 'relu')))
    model.add(TimeDistributed(Dense(1)))

    return model


def build_encoder_decoder_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    past_shape, future_shape, out_shape = get_io_shapes(data, params)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    recu_reg  = params['recu_reg']

    if len(out_shape) == 2:
      out_feats = out_shape[1]
    else:
      out_feats = 1

    # define training encoder
    past_inputs = Input(shape = past_shape,
                        name  = 'past_inputs')
    encoder = LSTM(feat_maps, return_state = True)
    encoder_outputs, enc_state_h, enc_state_c = encoder(past_inputs)
    encoder_states = [enc_state_h, enc_state_c]

    # define training decoder
    future_inputs = Input(shape = future_shape,
                          name  = 'future_inputs')
    decoder_lstm = LSTM(feat_maps,
                        return_sequences = True,
                        return_state     = True)
    x, _, _ = decoder_lstm(future_inputs, initial_state = encoder_states)
    dropout = Dropout(params['drop_out'])
    if params['drop_out'] != 0.0:
      x = dropout(x)
    decoder_dense_1 = Dense(16, activation = 'relu')
    decoder_dense_2 = Dense(1)
    decoder_dense_out = decoder_dense_1(x)
    decoder_outputs   = decoder_dense_2(decoder_dense_out)

    # define training encoder decoder
    ed_model = Model(inputs  = [past_inputs, future_inputs],
                     outputs = decoder_outputs,
                     name    = model_name)


    # define inference encoder
    models[model_name]['enc_model'] = Model(past_inputs,
```

```
                                              encoder_states,
                                              name = model_name + '_inf_enc')

    # define inference decoder
    decoder_future_inputs = Input(shape = (1, 1),
                                  name  = 'decoder_future_inputs')
    decoder_state_input_h = Input(shape = (feat_maps,))
    decoder_state_input_c = Input(shape = (feat_maps,))
    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
    x, dec_state_h, dec_state_c = decoder_lstm(decoder_future_inputs,
                                               initial_state = decoder_states_inpu
    decoder_states = [dec_state_h, dec_state_c]
    # No dropout at inference time
    decoder_dense_out = decoder_dense_1(x)
    decoder_outputs   = decoder_dense_2(decoder_dense_out)

    models[model_name]['dec_model'] = Model([decoder_future_inputs] + decoder_stat
                                            [decoder_outputs] + decoder_states,
                                            name = model_name + '_inf_dec')

    return ed_model


def build_autoencoder_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    past_shape, out_shape = get_io_shapes(data, params)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    recu_reg  = params['recu_reg']
    bs  = params['bs']

    if len(out_shape) == 2:
      out_feats = out_shape[1]
    else:
      out_feats = 1


    # input sequences
    enc_inputs = Input(shape = past_shape, name = 'enc_inputs')
    whole_sequence = LSTM(feat_maps, return_sequences = True)(enc_inputs)

    # Query-value attention of shape [batch_size, Tq, filters]
    query_value_attention_seq = Attention()([whole_sequence, whole_sequence])

    # build encoder model
    encoder = Model(enc_inputs, query_value_attention_seq, name = 'encoder')


    # input sequences - [batch_size, Tq, filters]
    dec_input = Input(shape=(past_shape[0], feat_maps), name = 'dec_inputs')
```

```
    whole_sequence = LSTM(feat_maps, return_sequences = True)(dec_input)

    # Query-value attention of shape [batch_size, Tq, filters]
    query_value_attention_seq = AdditiveAttention()([whole_sequence, dec_input])

    # Reduce over the sequence axis to produce shape [batch_size, filters]
    query_value_attention = GlobalAveragePooling1D()(query_value_attention_seq)

    # forecast
    #dense_out1 = Dense(feat_maps)(query_value_attention)
    #dec_output = Dense(out_steps)(dense_out1)
    dec_output = Dense(out_steps)(query_value_attention)

    # build decoder model
    decoder = Model(dec_input, dec_output, name = 'decoder')


    # encoder
    encoder_init   = Input(shape = past_shape)
    encoder_output = encoder(encoder_init)

    # decoder
    decoder_output = decoder(encoder_output)

    # autoencoder
    autoencoder = Model(encoder_init, decoder_output, name = model_name)

    return autoencoder


def get_model(models, params):

    if params['model_type'] == 'auto':
      model = build_autoencoder_model(models, params)
    elif params['model_type'] == 'encdec':
      model = build_encoder_decoder_model(models, params)
    elif params['model_type'] == 'simple':
      model = build_simple_encdec_model(models, params)

    return model


def get_default_params(model_type, steps = 48):
    params = {'xcols':        def_cols,
              'ycols':             'y',
              'lags':               48,
              'steps_ahead':     steps,
              'stride':          steps,
              'shuffle':          True,
              'bs':                 16,
              'model_type': model_type,
              'mix_type':         'ts',
              'mix_alpha':           4,
```

```
            'mix_factor':          0,
            'mix_diff':            1,
            'feat_maps':          32,
            'filters':             0,
            'kern_size':           0,
            'ks_feats':            0,
            'ks_time':             0,
            'drop_out':          0.0,
            'kern_reg':          0.0,
            'recu_reg':          0.0,
            'epochs':              5,
            'lrf_params': [0.00001, 10, 32, 5, 100, 25]}

    if params['model_type'] in ['transenc', 'transed', 'auto', 'encdec', 'simple']
      params.update({'lags': 24,
                     'bs':   32})

    if params['model_type'] in ['transed', 'transenc']:
      params.update({'feat_maps':    8,
                     'head_size':   64,
                     'num_heads':    4,
                     'trans_blocks': 2})

    return params


def run_model(models, params):
    model_name = get_model_name(models, params)

    h = compile_fit_validate(models, model_name, params)
    plot_history(h, model_name, params['epochs'])
    print_min_loss(h, model_name)

    return h
```

Define 2 related transformer-based MultiHeadAttention architectures:

- `build_transformer_encoder_decoder_model`
- `build_transformer_encoder_model`

```
# from tensorflow.keras import layers


class TransformerEncoder(Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super(TransformerEncoder, self).__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads

        self.attention = MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
```

```
        )

        self.dense_proj = keras.Sequential(
            [
                Dense(dense_dim, activation = "relu"),
                Dense(embed_dim),
            ]
        )

        self.layernorm_1 = LayerNormalization()
        self.layernorm_2 = LayerNormalization()


    def call(self, inputs, training):
        attention_output = self.attention(query = inputs,
                                          value = inputs,
                                          key   = inputs,
                                          training = training)
        proj_input  = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)

        return self.layernorm_2(proj_input + proj_output)


class PositionalEmbedding(Layer):
    def __init__(self, sequence_length, embed_dim, **kwargs):
        super(PositionalEmbedding, self).__init__(**kwargs)
        self.position_embeddings = Embedding(
            input_dim=sequence_length, output_dim=embed_dim
        )
        self.sequence_length = sequence_length
        self.embed_dim = embed_dim


    def call(self, inputs):
        length    = inputs.shape[1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_positions = self.position_embeddings(positions)

        return inputs + embedded_positions


    def compute_mask(self, inputs, mask = None):
        '''compute padding mask - currently unused'''
        # return tf.math.not_equal(inputs, 0)
        return tf.math.not_equal(inputs, tf.constant(-np.inf))


class TransformerDecoder(Layer):
    def __init__(self, embed_dim, latent_dim, num_heads, **kwargs):
        super(TransformerDecoder, self).__init__(**kwargs)
        self.embed_dim  = embed_dim
        self.latent_dim = latent_dim
        self.num_heads  = num_heads
```

```
        self.attention_1 = MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )

        self.attention_2 = MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )

        self.dense_proj = keras.Sequential(
            [
                Dense(latent_dim, activation = "relu"),
                Dense(embed_dim),
            ]
        )

        self.layernorm_1 = LayerNormalization()
        self.layernorm_2 = LayerNormalization()
        self.layernorm_3 = LayerNormalization()
        self.supports_masking = True


    def call(self, inputs, encoder_outputs, training, mask = None):
        causal_mask = self.get_causal_attention_mask(inputs)
        #if mask is not None:
        #    #print("mask:", mask)
        #    padding_mask = tf.cast(mask[:, tf.newaxis, :], dtype = "int32")
        #    #print("padding_mask:", padding_mask)
        #    #causal_mask = tf.minimum(padding_mask, causal_mask)
        #    #causal_mask = padding_mask & causal_mask
        #    causal_mask = padding_mask

        attention_output_1 = self.attention_1(query = inputs,
                                              value = inputs,
                                              key   = inputs,
                                              training = training,
                                              # use_causal_mask = True,  # 2.10 re
                                              attention_mask = causal_mask
        )
        out_1 = self.layernorm_1(inputs + attention_output_1)

        # mask = self.get_padding_mask(inputs, mask)
        # print("mask:", mask)

        if training == False:
          print("mask:", mask)
          padding_mask = tf.cast(mask[:, tf.newaxis, :], dtype = "int32")
          print("padding_mask:", padding_mask)

        #if mask is not None:
        #    #print("mask:", mask)
        #    padding_mask = tf.cast(mask[:, tf.newaxis, :], dtype = "int32")
        #    #print("padding_mask:", padding_mask)
        #    # padding_mask = tf.minimum(padding_mask, causal_mask)
```

```
    #    attention_output_2 = self.attention_2(query = out_1,
    #                                           value = encoder_outputs,
    #                                           key   = encoder_outputs,
    #                                           training = training,
    #                                           attention_mask = padding_mask
    #    )
    #else:
    attention_output_2 = self.attention_2(query = out_1,
                                          value = encoder_outputs,
                                          key   = encoder_outputs,
                                          training = training,
    )

    out_2 = self.layernorm_2(out_1 + attention_output_2)

    proj_output = self.dense_proj(out_2)

    return self.layernorm_3(out_2 + proj_output)


def get_causal_attention_mask(self, inputs):
    input_shape = tf.shape(inputs)
    batch_size, seq_length, num_feats = input_shape[0], input_shape[1], input_
    i = tf.range(seq_length)[:, tf.newaxis]
    j = tf.range(seq_length)
    mask = tf.cast(i >= j, dtype = "int32")
    mask = tf.reshape(mask, (1, seq_length, seq_length))
    mult = tf.concat(
        [tf.expand_dims(batch_size, -1), tf.constant([1, 1], dtype = tf.int32)
        axis = 0,
    )

    return tf.tile(mask, mult)


def get_padding_mask(self, inputs, mask = None):
    '''Compute padding mask

    np.inf can be used at inference time to initialise the decoder input
    (bs, seq_length, num_features).  Cannot use 0 or 0.0 because these
    are valid (scaled and centered) temperatures.  Positions in the decoder
    input which equal np.inf values get masked in the decoder
    cross-attention layer.

    The decoder predictions replace np.inf values and are fed back into
    the decoder.  The decoder predicts the next value based on the previous
    values it predicted.

    In NLP, padding is more commonly used in the encoder for variable
    length sequences.  That isn't relevant with this time series.

    '''

    inf_bools = tf.math.not_equal(inputs, tf.constant(-np.inf))
```

```
        inf_ints  = tf.cast(inf_bools, dtype = "int32")
        inf_sum   = tf.reduce_sum(inf_ints)

        print("reduce_sum:", inf_sum)

        if tf.math.equal(inf_sum, tf.constant(0)):
          res = mask
        else:
          res = inf_bools

        return res


def build_transformer_encoder_decoder_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    past_shape, future_shape, out_shape = get_io_shapes(data, params)
    out_steps = out_shape[0]

    # print("past_shape:", past_shape)
    # print("future_shape:", future_shape)
    # print("out_shape:", out_shape)

    if len(out_shape) == 2:
      out_feats = out_shape[1]
    else:
      out_feats = 1

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    embed_dim = params['head_size']
    num_heads = params['num_heads']
    trans_blocks = params['trans_blocks']


    ##########################################################################
    # encoder
    encoder_inputs = Input(shape = past_shape, name = 'enc_inputs')
    enc_seq_length = encoder_inputs.shape[1]

    x_enc = PositionalEmbedding(enc_seq_length, embed_dim)(encoder_inputs)

    for _ in range(trans_blocks):
        x_enc = TransformerEncoder(embed_dim, feat_maps, num_heads)(inputs   = x_e
                                                                    training = Tru
    encoder_outputs = x_enc

    encoder = Model(encoder_inputs,
                    encoder_outputs,
                    name = model_name + '_inf_enc')

    models[model_name]['enc_model'] = encoder
```

```
    ##########################################################################
    # decoder
    decoder_inputs = Input(shape = future_shape, name = 'dec_inputs')
    dec_seq_length = decoder_inputs.shape[1]
    # decoder_inputs = Input(shape = (None, 1), name = 'dec_inputs')  # set timest
    # dec_seq_length = future_shape[1]
    encoded_seq_inputs = Input(shape = (None, embed_dim),
                               name = "decoder_state_inputs")

    x_dec = PositionalEmbedding(dec_seq_length, embed_dim)(decoder_inputs)

    for _ in range(trans_blocks):
        x_dec = TransformerDecoder(embed_dim, feat_maps, num_heads)(inputs = x_dec
                                                                    encoder_output
                                                                    training = Tru

    # x_dec = Dropout(0.1)(x_dec)

    x_dec = Dense(64, activation = 'relu')(x_dec)
    decoder_outputs = Dense(1)(x_dec)

    decoder = Model([decoder_inputs, encoded_seq_inputs],
                    decoder_outputs,
                    name = model_name + '_inf_dec')

    models[model_name]['dec_model'] = decoder


    ##########################################################################
    # transformer
    decoder_outputs = decoder([decoder_inputs, encoder_outputs])
    transformer = Model([encoder_inputs, decoder_inputs],
                        decoder_outputs,
                        name = model_name)

    return transformer


def build_transformer_encoder_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    past_shape, out_shape = get_io_shapes(data, params)
    out_steps = out_shape[0]

    # print("past_shape:", past_shape)
    # print("out_shape:", out_shape)

    if len(out_shape) == 2:
      out_feats = out_shape[1]
    else:
      out_feats = 1

    feat_maps = params['feat_maps']
```

```
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    embed_dim = params['head_size']
    num_heads = params['num_heads']
    trans_blocks = params['trans_blocks']


    ###########################################################################
    # encoder(s)
    encoder_inputs = Input(shape = past_shape, name = 'enc_inputs')
    enc_seq_length = encoder_inputs.shape[1]

    x_enc = PositionalEmbedding(enc_seq_length, embed_dim)(encoder_inputs)

    for _ in range(trans_blocks):
        x_enc = TransformerEncoder(embed_dim, feat_maps, num_heads)(inputs   = x_e
                                                                      training = Tru

    # x_enc = Dropout(drop_out)(x_enc)

    x_enc = Dense(feat_maps, activation = 'relu')(x_enc)
    encoder_outputs = Dense(1)(x_enc)

    # transformer
    transformer = Model(encoder_inputs,
                        encoder_outputs,
                        name = model_name)

    return transformer


def get_model(models, params):

    if params['model_type'] == 'transenc':
      model = build_transformer_encoder_model(models, params)
    elif params['model_type'] == 'transed':
      model = build_transformer_encoder_decoder_model(models, params)
    elif params['model_type'] == 'auto':
      model = build_autoencoder_model(models, params)
    elif params['model_type'] == 'encdec':
      model = build_encoder_decoder_model(models, params)
    elif params['model_type'] == 'simple':
      model = build_simple_encdec_model(models, params)

    return model
```

Specify some utility functions for running, plotting and summarising results:

- `plot_history`
- `plot_forecasts`
- `plot_horizon_metrics`
- `check_residuals`

For running multiple models with specified parameters:

- `random_search_params` - multiple parameters eg. lags and feature_maps
- `sweep_param` - single parameter eg. lags

and summarising performance of multiple models:

- `rank_models`
- `get_best_models`

Note that I don't use the `random_search_params` function all that much in this notebook because I prefer the scikit-optimize approach outlined in the code cell following this one.

```
def compile_fit_validate(models, model_name, params, verbose = 2):
    # Reduces variance in results but won't eliminate it :-(
    random.seed(42)
    np.random.seed(42)
    tf.random.set_seed(42)

    model = models[model_name]['model']
    train_data = models[model_name]['train']
    valid_data = models[model_name]['valid']

    model.summary()

    # opt = Adam(learning_rate = 0.001)
    opt = Adam(models[model_name]['lrf'].best_lr)

    model.compile(optimizer = opt, loss = 'mse', metrics = ['mae'])

    es = EarlyStopping(monitor = 'val_loss',
                       mode = 'min',
                       verbose = 1,
                       patience = 10,
                       restore_best_weights = True)  # return best model, not last
    lr = ReduceLROnPlateau(monitor = 'val_loss',
                           factor = 0.2,
                           patience = 5,
                           min_lr = 0.00001)

    h = model.fit(train_data, validation_data = valid_data,
                  epochs = params['epochs'], verbose = verbose, callbacks = [es, l

    return h


def plot_history(h, name, epochs = 10):
    fig, axs = plt.subplots(1, 2, figsize = (9, 6), tight_layout = True)
    axs = axs.ravel()

    if 'fm_' in name:
      name = name.replace('fm_', 'fm\n')

    axs[0].plot(h.history['loss'])
```

```
    axs[0].plot(h.history['val_loss'])
    axs[0].set_title(name + '\nloss')
    axs[0].set_xticklabels(range(1, epochs + 1))
    axs[0].set_xticks(range(0, epochs))
    axs[0].set_ylabel('loss')
    axs[0].set_xlabel('epoch')
    axs[0].legend(['train', 'valid'], loc = 'upper right')

    axs[1].plot(h.history['mae'])
    axs[1].plot(h.history['val_mae'])
    axs[1].set_title(name + '\nmae')
    axs[1].set_xticks(range(0, epochs))
    axs[1].set_xticklabels(range(1, epochs + 1))
    axs[1].set_ylabel('mae')
    axs[1].set_xlabel('epoch')
    axs[1].legend(['train', 'valid'], loc = 'upper right')
    plt.show()

    return None


def print_min_loss(h, name):
    argmin_loss     = np.argmin(np.array(h.history['loss']))
    argmin_val_loss = np.argmin(np.array(h.history['val_loss']))
    min_loss        = h.history['loss'][argmin_loss]
    min_val_loss    = h.history['val_loss'][argmin_val_loss]
    mae             = h.history['mae'][argmin_loss]
    val_mae         = h.history['val_mae'][argmin_val_loss]

    txt = "{0:s} {1:s} min loss: {2:f}\tmae: {3:f}\tepoch: {4:d}"
    print(txt.format(name, "train", min_loss,     mae,     argmin_loss + 1))
    print(txt.format(name, "valid", min_val_loss, val_mae, argmin_val_loss + 1))
    print()

    return None


def plot_forecasts(models, model_name, dataset = 'valid', subplots = 3):
    """Plot example forecasts with observations and lagged temperatures.

       First row shows near zero rmse forecasts.
       Second row shows most positive rmse forecasts.
       Third row shows most negative rmse forecasts.
    """

    # get model etc
    model   = models[model_name]['model']
    params  = models[model_name]['params']
    horizon = params['steps_ahead']
    lags    = params['lags']

    assert horizon >= 12
    assert subplots in [3, 4, 5]
```

```
# get data
if dataset == 'test':
  data = models[model_name]['test']
elif dataset == 'train':
  data = models[model_name]['train']
elif dataset == 'valid':
  data = models[model_name]['valid']
else:
  print("Unknown dataset:", dataset)
  return None

# make forecast
preds = model.predict(data)
preds = preds.reshape((preds.shape[0], preds.shape[1]))
preds = preds[::horizon]

obs   = np.concatenate([y for _, y in data], axis = 0)
long_obs = obs.reshape((obs.shape[0], obs.shape[1]))
long_obs = long_obs[::horizon]

res = long_obs - preds  # res for residual
res_sign = np.sign(-res.mean(axis = 1))

err = (long_obs - preds) ** 2  # err for error
err_row_means = err.mean(axis = 1)
rmse_rows = res_sign * np.sqrt(err_row_means)

# choose forecasts
neg_rmse = np.argsort(rmse_rows)[:subplots]
pos_rmse = np.argsort(-rmse_rows)[:subplots]
nz_rmse  = np.argsort(np.abs(rmse_rows))[:subplots]  # nz near zero

plot_idx = np.concatenate((nz_rmse, pos_rmse, neg_rmse))

# plot forecasts
fig, axs = plt.subplots(3, subplots, sharex = True, sharey = True, figsize = (
axs = axs.ravel()

for i in range(3 * subplots):
  lagged_obs = get_lagged_obs(long_obs, plot_idx[i] - 1, lags)
  axs[i].plot(range(-lags + 1, 1),
              inv_transform(scaler, lagged_obs, 'y', models['datasets']['trair
              'blue',
              label='lagged observations')
  axs[i].plot(range(1, horizon + 1),
              inv_transform(scaler, preds[plot_idx[i]],    'y', models['datase
              'orange',
              label='forecast')
  axs[i].plot(range(0, horizon),
              inv_transform(scaler, long_obs[plot_idx[i]], 'y', models['datase
              'green',
              label='observations')
  sub_title = "{0:d} {1:.4f}".format(plot_idx[i], rmse_rows[plot_idx[i]])
  axs[i].title.set_text(sub_title)
```

```
    fig.suptitle(model_name + " " + dataset + "\nperiod idx, signed rmse")
    fig.text(0.5, 0.04, 'forecast horizon - half hour steps', ha='center')
    fig.text(0.04, 0.5, 'Temperature - $^\circ$C', va='center', rotation='vertical
    plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show();


def get_lagged_obs(long_obs, plot_idx, lags):
    if long_obs[plot_idx].size < lags:
      lagged_obs = np.flip(long_obs[plot_idx])
    else:
      lagged_obs = long_obs[plot_idx]

    while lagged_obs.size < lags:
      plot_idx -= 1
      lagged_obs = np.concatenate([lagged_obs, np.flip(long_obs[plot_idx])])

    if long_obs[plot_idx].size < lags:
      lagged_obs = np.flip(lagged_obs)

    return lagged_obs[-lags:]


def plot_horizon_metrics(models, model_name, dataset = 'valid'):

    # get model etc
    model   = models[model_name]['model']
    params  = models[model_name]['params']
    horizon = params['steps_ahead']

    assert horizon >= 12

    # get data
    if dataset == 'test':
      data = models[model_name]['test']
    elif dataset == 'train':
      data = models[model_name]['train']
    elif dataset == 'valid':
      data = models[model_name]['valid']
    else:
      print("Unknown dataset:", dataset)
      return None

    # make forecast
    if params['model_type'] in ['encdec', 'transed']:
      obs, preds = predict_encoder_decoder_sequence(models, params, dataset)
    else:
      preds = model.predict(data)
      obs   = np.concatenate([y for _, y in data], axis = 0)

    if len(obs.shape) == 3 and len(preds.shape) == 3:
      # multi-step, multi-feature output
      preds = preds[:, :, 0:1]
```

```
      preds = preds.reshape((preds.shape[0], preds.shape[1]))
      obs = obs[:, :, 0:1]
      obs = obs.reshape((obs.shape[0], obs.shape[1]))
    elif len(obs.shape) == 3 and len(preds.shape) == 2:
      obs = obs.reshape((obs.shape[0], obs.shape[1]))

    assert preds.shape == obs.shape

    # calculate metrics
    rmse_h, mae_h = np.zeros(horizon), np.zeros(horizon)

    for i in range(horizon):
      t_obs   = inv_transform(scaler, obs[:, i],   'y', models['datasets']['train
      t_preds = inv_transform(scaler, preds[:, i], 'y', models['datasets']['train
      rmse_h[i] = rmse(t_obs, t_preds)
      mae_h[i]  = np.median(np.abs(t_obs - t_preds))  # for comparison with basel:
      #mae_h[i]  = mae(t_obs,  t_preds)

    # plot metrics for horizons
    fig, axs = plt.subplots(1, 2, figsize = (14, 7))
    fig.suptitle(model_name + " " + dataset)
    axs = axs.ravel()

    mean_val_lab = model_name + ' mean value'
    axs[0].plot(range(1, horizon+1), rmse_h, label=model_name)
    if dataset == 'test':
      var_rmse = np.array([0.39, 0.52, 0.64, 0.75, 0.86, 0.96, 1.06, 1.15, 1.23, 1
       1.45, 1.51, 1.57, 1.63, 1.68, 1.73, 1.77, 1.81, 1.85, 1.89, 1.92,
       1.96, 1.99, 2.02, 2.05, 2.08, 2.1 , 2.13, 2.15, 2.18, 2.2 , 2.22,
       2.24, 2.26, 2.28, 2.3 , 2.31, 2.33, 2.35, 2.36, 2.38, 2.39, 2.4 ,
       2.42, 2.43, 2.44, 2.45])
      axs[0].plot(range(1, horizon+1), var_rmse, label='VAR')
    else:
      axs[0].hlines(np.mean(rmse_h), xmin=1, xmax=horizon, color='yellow', linesty
    axs[0].set_xlabel("horizon - half hour steps")
    axs[0].set_ylabel("rmse")

    axs[1].plot(range(1, horizon+1), mae_h, label=model_name)
    if dataset == 'test':
      var_mae = np.array([0.39, 0.49, 0.57, 0.66, 0.74, 0.83, 0.91, 0.98, 1.05, 1.
       1.24, 1.29, 1.34, 1.39, 1.43, 1.47, 1.5 , 1.53, 1.56, 1.59, 1.62,
       1.64, 1.66, 1.68, 1.7 , 1.72, 1.73, 1.75, 1.76, 1.77, 1.78, 1.8 ,
       1.81, 1.82, 1.83, 1.83, 1.84, 1.85, 1.85, 1.86, 1.86, 1.87, 1.87,
       1.88, 1.88, 1.89, 1.89])
      axs[1].plot(range(1, horizon+1), var_mae, label='VAR')
    else:
      axs[1].hlines(np.mean(mae_h), xmin=1, xmax=horizon, color='yellow', linestyl
    axs[1].set_xlabel("horizon - half hour steps")
    axs[1].set_ylabel("mae")
    plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()


def plot_obs_preds(obs, preds, title):
```

```
    plt.figure(figsize = (12, 8))
    plt.subplot(3, 1, 1)
    plt.scatter(x = obs, y = preds)
    y_lim = plt.ylim()
    x_lim = plt.xlim()
    plt.plot(x_lim, y_lim, 'k-', color = 'grey')
    plt.xlabel('Observations')
    plt.ylabel('Predictions')
    plt.title(title)


def plot_residuals(obs, preds, title):
    plt.subplot(3, 1, 2)
    plt.scatter(x = range(len(obs)), y = (obs - preds))
    plt.axhline(y = 0, color = 'grey')
    plt.xlabel('Position')
    plt.ylabel('Residuals')
    plt.title(title)


def plot_residuals_dist(obs, preds, title):
    data = obs - preds
    plt.subplot(3, 1, 3)
    pd.Series(data).plot(kind = 'density')
    plt.axvline(x = 0, color = 'grey')
    plt.title(title)
    plt.tight_layout()
    plt.show()


def check_residuals(models, model_name, dataset = 'valid'):
    """Plot observations against predictions, residuals and residual distribution

    Warning: The full training set will take approx. 5 mins to plot"""

    assert dataset in ['test', 'valid']

    model = models[model_name]
    data  = model[dataset]

    if model['params']['model_type'] in ['encdec', 'transed']:
      obs, preds = predict_encoder_decoder_sequence(models,
                                                    models[model_name]['params'],
                                                    dataset)
    else:
      preds = model['model'].predict(data)
      obs   = np.concatenate([y for _, y in data], axis = 0)

    # reshape obs & preds
    label_len = obs.shape[0]
    preds_len = len(preds)
    # print("labels:", label_len)
    # print("preds:",  preds_len)
    # print("preds:",  preds.shape)
```

```
    # print("obs:",    obs.shape)
    assert label_len == preds_len

    # print("obs[0]:", obs.shape[0])
    # print("obs[1]:", obs.shape[1])
    preds_long = preds.reshape((obs.shape[0] * obs.shape[1]))
    obs_long   = obs.reshape((obs.shape[0] * obs.shape[1]))

    mse_long = mse(obs_long, preds_long)  # Need to treat 4 step ahead rmse & mae
    mae_long = mae(obs_long, preds_long)
    print("mse ", model_name, ": ",  mse_long, sep = '')
    print("mae ", model_name, ":  ", mae_long, sep = '')

    # inverse transform using train mean & sd
    t_preds = inv_transform(scaler, preds_long, 'y', train_df.columns)
    t_obs   = inv_transform(scaler, obs_long,   'y', train_df.columns)

    t_rmse = rmse(t_obs, t_preds)  # Need to treat 4 step ahead rmse & mae proper
    t_mae  =  mae(t_obs, t_preds)
    print("t rmse ", model_name, ": ",  t_rmse, sep = '')
    print("t mae ",  model_name, ":  ", t_mae,  sep = '')

    title = 'Inverse transformed data\n' + model_name
    plot_obs_preds(t_obs, t_preds, title)
    plot_residuals(t_obs, t_preds, title)
    plot_residuals_dist(t_obs, t_preds, title)
    print("\n\n")


def rmse(obs, preds):
    return np.sqrt(np.mean((obs - preds) ** 2))


def mse(obs, preds):
    return np.mean((obs - preds) ** 2)


def mae(obs, preds):
    "mean absolute error - equivalent to the keras loss function"
    return np.mean(np.abs(obs - preds))  # keras loss
    # return np.median(np.abs(obs - preds))  # earlier baselines


def predict_encoder_decoder_sequence(models, params, dataset = 'valid', samples =
    """Make predictions for encoder decoder models

    Iterates through all the batches in the dataset

    Args:
      models     (dict):             all models
      params     (dict):             parameters of model of interest
      dataset    (str, optional):   test, train or validate dataset
      samples    (int, optional):   number of regularly spaced samples to run fro
```

```
Returns:
  obs        (pd.DataFrame)
  preds      (pd.DataFrame)

Notes: ...
"""

assert params['model_type'] in ['encdec', 'transed']

model_name = get_model_name(models, params)
preds_name = dataset + '_preds'
obs_name   = dataset + '_obs'

# get models
infenc = models[model_name]['enc_model']
infdec = models[model_name]['dec_model']
infmod = infdec = models[model_name]['model']

# get data
if dataset == 'test':
  data = models[model_name]['test']
elif dataset == 'train':
  data = models[model_name]['train']
elif dataset == 'valid':
  data = models[model_name]['valid']
else:
  print("Unknown dataset:", dataset)
  return None

# if preds_name in models[model_name] and obs_name in models[model_name]:
#   return models[model_name][obs_name], models[model_name][preds_name]

# obs = np.concatenate([y for _, y in data], axis = 0)

# return obs and preds instead?
# mse_  = list()
# mae_  = list()
obs   = list()
preds = list()

# get predictions
i = 0
interval = 1
if samples != None:
  batches  = data.cardinality().numpy()
  interval = batches // samples

for batch in data:
  i += 1

  if i % interval != 0:
    continue

  #if i >= data.cardinality().numpy():
```

```
        #  continue

        past = batch[0][0]
        obs_ = batch[1].numpy()
        obs_ = obs_.squeeze()
        obs.append(obs_)

        if params['model_type'] == 'encdec':
            preds_ = predict_sequence(infenc, infdec, past, params)
        elif params['model_type'] == 'transed':
            # preds_ = predict_sequence_transformer(infenc, infdec, past, params)
            preds_ = predict_sequence_transformer(infenc, infmod, past, params)
        preds.append(preds_)

        # plt.plot(range(-params['lags'], 0),    past[0, :, 0])
        # plt.plot(range(params['steps_ahead']), obs_)
        # plt.plot(range(params['steps_ahead']), preds_)
        # plt.show()

        # summarise predictions
        # mse_.append(mse(obs_, preds_))
        # mae_.append(mae(obs_, preds_))

    # print("mse:", np.mean(mse_))
    # print("mae:", np.mean(mae_))

    preds = np.concatenate(preds, axis=0)
    obs   = np.concatenate(obs,   axis=0)
    models[model_name][preds_name] = preds
    models[model_name][obs_name]   = obs

    return obs, preds

def compute_mask(inputs, mask = None):
    '''compute padding mask - currently unused'''
    # return tf.math.not_equal(inputs, 0)
    return tf.math.not_equal(inputs, tf.constant(-np.inf))


def predict_sequence_transformer(infenc, infmod, source, params):
    """Generate target given source sequence

    Operates on batches of data

    Args:
      infenc    (Keras model):      inference encoder model
      infmod    (Keras model):      inference model
      source    (Keras data batch): input data
      params    (dict):             parameters of model of interest

    Returns:
      output    (pd.DataFrame):    output target sequence

    Notes: based on
```

```
        https://machinelearningmastery.com/develop-encoder-decoder-model-sequer
"""

bs = source.shape[0]
n_steps = params['steps_ahead']

# encode
#state = infenc.predict(source)  # Not used!?
# start of sequence input
# print("source:", source.shape)
#source_len = source.shape[1] - 1
target_seq = source[:, -n_steps:, 0]  #  working :-)
target_seq = np.array(target_seq).reshape(bs, n_steps, 1)  # working :-)
#target_seq = source[:, -1, 0]  #  NEXT try this?
#target_seq = np.array(target_seq).reshape(bs, 1, 1)
#target_seq = np.array([-np.inf] * bs * n_steps).reshape(bs, n_steps, 1) # was
#for i in range(bs):
#  target_seq[i, 0, 0] = source[i, -1, 0]
#target_seq = source[:, -1, 0]  #  terrible!
#target_seq = np.array(target_seq).reshape(bs, 1, 1)  # terrible!
# target_seq = source[:, :, 0]
# target_seq = np.array([source[i][source_len][0] for i in range(bs)]).reshape
# target_seq = np.array([[-np.inf] * n_steps for i in range(bs)]).reshape(bs,
# target_seq = np.array([[False] * n_steps for i in range(bs)]).reshape(bs, n_
# target_seq = np.array([source[i][source_len - n_steps:source_len][0] for i i
#target_seq = np.array([source[i][source_len][0] for i in range(bs)]).reshape(

# collect predictions
output = list()

# print("source:", source.shape)
# print("target_seq:", target_seq.shape)

for t in range(1):
  # print("source:", source.shape)
  #print("target_seq:", target_seq.shape)
  # yhat, h, c = infmod.predict([target_seq] + state)  # encdec
  # yhat = infmod.predict([target_seq, state])  # transed ([decoder_inputs, en
  #padding_mask = compute_mask(target_seq)
  #yhat = infmod([state, target_seq], training = False, mask = padding_mask)
  # yhat = infmod.predict([np.array(source), np.array(target_seq)])
  # yhat = infmod.predict([source, target_seq])  # transformer - working :-)
  yhat = infmod([source, target_seq], training = False, mask = None)  # new wo
  #print("yhat:", yhat.shape)
  #if t < 5:
  #  print("t:", t)
  #  print("target_seq[0, :, 0]:", target_seq[0, :, 0])
  #  print("yhat[0, :, 0]:", yhat[0, :, 0])
  #  #print("padding_mask[0, :, 0]:", padding_mask[0, :, 0])
  #else:
  #  foobar()
  # store prediction
  # output.append(yhat[:, 0, 0])   # store 1st prediction
  #output.append(yhat[:, t, 0])     # store tth prediction  - getting better
```

```
        #output.append(yhat[:, -1, 0])  # store last prediction - working :-)
        output = yhat
        # update state
        # state = [h, c]  # encdec
        # update target sequence
        # target_seq = yhat  # encdec
        # target_seq = np.append(target_seq, yhat)
        #target_seq = yhat[:, -1, 0].reshape(bs, 1, 1)  # terrible!
        #target_seq = target_seq[:, 1:, 0].reshape(bs, n_steps-1, 1)  # working :-)
        # print("target_seq:", target_seq.shape)
        #target_seq = np.append(target_seq, yhat[:, n_steps-1, 0]).reshape(bs, n_ste
        #target_seq = np.concatenate([target_seq, yhat[:, t, 0, None, None]], axis=1
        #target_seq = np.concatenate([target_seq, yhat[:, -1, 0, None, None]], axis=
        #target_seq = np.insert(target_seq, t, yhat[:, -1, 0, None, None], axis=1)
        #for i in range(bs):
        #  target_seq[i, t, 0] = yhat[i, t, 0]  # getting better
        # target_seq = np.array([target_seq[i, n_steps, 0] = yhat[i, n_steps-1, 0] f
        #for i in range(bs):
        #  target_seq = np.concatenate(target_seq[i, n_steps-2, 0], yhat[i, n_steps-
        #target_seq = np.vstack([target_seq[:, :, 0], yhat[:, n_steps-1, 0]])

    #return np.array(output).transpose()
    return output


def predict_sequence(infenc, infdec, source, params):
    """Generate target given source sequence

    Operates on batches of data

    Args:
      infenc    (Keras model):      inference encoder model
      infdec    (Keras model):      inference decoder model
      source    (Keras data batch): input data
      params    (dict):             parameters of model of interest

    Returns:
      output    (pd.DataFrame):    output target sequence

    Notes: based on
           https://machinelearningmastery.com/develop-encoder-decoder-model-sequen
    """

    bs = source.shape[0]
    n_steps = params['steps_ahead']

    # encode
    state = infenc.predict(source)
    # start of sequence input
    source_len = source.shape[1] - 1
    target_seq = np.array([source[i][source_len][0] for i in range(bs)]).reshape(b
    # collect predictions
    output = list()
```

```
    for t in range(n_steps):
      # next prediction
      yhat, h, c = infdec.predict([target_seq] + state)  # encdec
      # store prediction
      output.append(yhat[:, 0, 0])
      # update state
      state = [h, c]
      # update target sequence
      target_seq = yhat

    return np.array(output).transpose()


def expand_grid(dictionary):
   return pd.DataFrame([row for row in product(*dictionary.values())],
                       columns = dictionary.keys())


def random_search_params(models, params, sweep_values, limit = 5):
    sweep_params = list(sweep_values.keys())
    assert len(sweep_params) > 1

    i = 0
    model_names = []
    sweep_df    = expand_grid(sweep_values)
    sweep_rows = sweep_df.sample(n = limit)

    for sweep_row in sweep_rows.itertuples():
      i += 1
      print("%d of %d" %(i, limit))
      print(sweep_row)
      for idx in sweep_params:
        params[idx] = getattr(sweep_row, idx)

      model_name = get_model_name(models, params)
      model_names.append(model_name)
      models[model_name] = {}
      models[model_name]['params'] = params

      ds_train, ds_valid, ds_test = make_datasets(models, params)
      models[model_name]['train'] = ds_train
      models[model_name]['valid'] = ds_valid
      models[model_name]['test']  = ds_test

      models[model_name]['model'] = get_model(models, params)
      models[model_name]['lrf']   = run_lrf(models, params)
      models[model_name]['history'] = run_model(models, params)

    summarise_history(models, model_names)

    return [models, model_names]


def sweep_param(models, params, sweep_values, verbose=False):
```

```
    sweep_params = list(sweep_values.keys())
    sweep_param  = sweep_params[0]
    assert len(sweep_params) == 1
    assert len(sweep_values[sweep_param]) >= 1

    model_names = []

    for sweep_value in sweep_values[sweep_param]:
      # params_copy = {key: value[:] for key, value in params.items()}
      params_copy = {key: value for key, value in params.items()}
      params_copy[sweep_param] = sweep_value

      if verbose == True:
        print(sweep_param, ":", sweep_value)

      model_name = get_model_name(models, params_copy)
      model_names.append(model_name)
      models[model_name] = {}
      models[model_name]['params'] = params_copy

      ds_train, ds_valid, ds_test = make_datasets(models, params_copy)
      models[model_name]['train'] = ds_train
      models[model_name]['valid'] = ds_valid
      models[model_name]['test']  = ds_test

      models[model_name]['model']   = get_model(models, params_copy)
      models[model_name]['lrf']     = run_lrf(models,   params_copy)
      models[model_name]['history'] = run_model(models, params_copy)

    summarise_history(models, model_names)

    return [models, model_names]


def check_fit(h, metric, fit_type, ignore = 1):
    badfit = 0

    h_train = h.history[metric]
    h_valid = h.history['val_' + metric]
    h_len = len(np.array(h_train))

    for i in range(ignore, h_len):
      # Disabling underfitting check for now
      # if ( fit_type == 'over'  and h_valid[i] < h_train[i] ) or \
      #    ( fit_type == 'under' and h_valid[i] > h_train[ignore] ):
      if ( fit_type == 'over'  and h_valid[i] < h_train[i] ):
        badfit += 1

    return round(badfit * 100 / (h_len - ignore), 2)


def get_history_stats(h, metric, ignore = 0):
    stats = {}
```

```
    stats['mean'] = np.mean(np.array(h.history[metric]))
    stats['std']  = np.std(np.array(h.history[metric]))

    h_argmin = np.argmin(np.array(h.history[metric]))
    h_argmax = np.argmax(np.array(h.history[metric]))
    stats['min'] = h.history[metric][h_argmin]
    stats['max'] = h.history[metric][h_argmax]
    stats['argmin'] = h_argmin

    h_len = len(np.array(h.history[metric]))
    stats['first'] = np.array(h.history[metric])[0]
    stats['last']  = np.array(h.history[metric])[h_len - 1]

    # monotonically decreasing
    stats['monod'] = np.all(np.diff(h.history[metric]) < 0)

    stats['max_eq_first'] = stats['max'] == stats['first']
    stats['min_eq_last']  = stats['min'] == stats['last']

    return stats


def summarise_history(models, model_names):

    for model_name in model_names:
      if model_name == '':
        continue

      model = models[model_name]
      model['perf'] = {}
      mod_perf = model['perf']
      mod_perf['val_loss'] = get_history_stats(model['history'], 'val_loss')
      mod_perf['val_mae']  = get_history_stats(model['history'], 'val_mae')

      mod_perf['loss'], mod_perf['mae'] = {}, {}
      mod_perf['loss']['overfit_pc']  = check_fit(model['history'], 'loss', 'over
      mod_perf['loss']['underfit_pc'] = check_fit(model['history'], 'loss', 'unde
      mod_perf['mae']['overfit_pc']   = check_fit(model['history'], 'mae',  'over
      mod_perf['mae']['underfit_pc']  = check_fit(model['history'], 'mae',  'unde

    return None


def get_all_model_names(models):
    names = []

    for name in models.keys():
      if not name in ['datasets']:
        names.append(name)

    return names


def reject_model(mod_perf, strict):
```

```
    fit_pc_lim = 0.0
    reject = False

    if mod_perf['loss']['overfit_pc']  > fit_pc_lim or \
       mod_perf['loss']['underfit_pc'] > fit_pc_lim or \
       (strict == True and mod_perf['mae']['overfit_pc']   > fit_pc_lim) or \
       (strict == True and mod_perf['mae']['underfit_pc']  > fit_pc_lim):
      reject = True

    if (strict == True and mod_perf['val_loss']['monod'] == False) or \
       (strict == True and mod_perf['val_mae']['monod']  == False):
      reject = True

    return reject


def get_best_models(models, model_names = None, strict = False):
    best_mse_mod, best_mae_mod = None, None
    low_mse, low_mae = sys.maxsize, sys.maxsize

    if model_names == None:
      model_names = get_all_model_names(models)

    for model_name in model_names:
      model = models[model_name]

      try:
        mod_perf = model['perf']
      except:
        continue

      if reject_model(mod_perf, strict):
        continue

      if mod_perf['val_loss']['min'] < low_mse:
        low_mse = mod_perf['val_loss']['min']
        best_mse_mod = model_name

      if mod_perf['val_mae']['min'] < low_mae:
        low_mae = mod_perf['val_mae']['min']
        best_mae_mod = model_name

    return ['low mse ' + str(best_mse_mod), round(low_mse, 5),
            'low mae ' + str(best_mae_mod), round(low_mae, 5)]


def plot_perf_boxplot(models, metric, model_names = None, strict = False):
    stats = []

    assert metric in ['val_loss', 'val_mae']

    if model_names == None:
      model_names = get_all_model_names(models)
      title = 'All models'
```

```
    else:
      # title = [k for k, v in locals().items() if v == 'model_names']
      title = str(len(model_names)) + ' models'

    title += ' - strict=' + str(strict)

    for model_name in model_names:
      try:
        mod_perf = models[model_name]['perf']
      except:
        continue

      if reject_model(mod_perf, strict):
        continue

      stats.append(mod_perf[metric]['min'])

    assert len(stats) > 2

    fig1, ax1 = plt.subplots()
    ax1.set_title(title + ' ' + metric)
    ax1.boxplot(stats, labels=['']);


def rank_models(models, metric, model_names = None, strict = False, limit = 5):
    stats = {}

    assert metric in ['val_loss', 'val_mae']

    if model_names == None:
      model_names = get_all_model_names(models)

    for model_name in model_names:
      try:
        mod_perf = models[model_name]['perf']
      except:
        continue

      if reject_model(mod_perf, strict):
        continue

      stats[model_name] = round(mod_perf[metric]['min'], 5)

    return sorted(stats.items(), key=lambda item: item[1])[:limit]
    # return [dict(sorted(stats.items(), key=lambda item: item[1]))][:limit]


def keep_key(d, k):
  """ models = keep_key(models, 'datasets') """
  return {k: d[k]}
```

Bayesian hyperparameter optimization

I've used the BayesianOptimization package in the past to optimise time series forecasts. It works well but doesn't have any plotting functions. It should be possible to spot irrelevant hyperparameters with the scikit-optimize plot_objective function even if the underlying gaussian processes are approximations.

The `model_fitness_1s` example function is passed to `gp_minimize` from scikit-optimize. The `model_fitness_1s` function should be seen as an implementation example which will be customised later for particular network architectures and parameters to optimise.

```
# skopt now available on colab :-)
# !pip freeze
# !pip install scikit-optimize

import skopt
from skopt import gp_minimize
from skopt.space import Real, Categorical, Integer
from skopt.plots import plot_convergence, plot_objective, plot_evaluations, \
                        plot_gaussian_process
from skopt.utils import use_named_args

print("\nskopt version:", skopt.__version__)


dim_lags = Integer(low = 4,  high = 48, name = 'lags')
dim_bs   = Integer(low = 16, high = 32, name = 'bs')
dim_fm   = Integer(low = 16, high = 32, name = 'feat_maps')
dim_drop_out = Real(low = 1e-3, high = 5e-1, prior = 'log-uniform', name = 'drop_c

bo_dims_1s = [dim_lags,
              dim_bs,
              dim_fm,
              dim_drop_out]


def create_model(params):

    model_name = get_model_name(models, params)
    models[model_name] = {}
    models[model_name]['params'] = params

    ds_train, ds_valid, ds_test = make_datasets(models, params)
    models[model_name]['train'] = ds_train
    models[model_name]['valid'] = ds_valid
    models[model_name]['test']  = ds_test

    models[model_name]['model'] = get_model(models, params)
    models[model_name]['lrf']   = run_lrf(models, params)

    return models[model_name]['model']


def get_bo_mse(params, **dims):
```

```
    params.update(**dims)

    for k, v in dims.items():
        print(k, v)

    model_names = ['']
    model_name = get_model_name(models, params)
    model_names.append(model_name)

    # skopt will re-evaluate the same point, even when gp_minimize(..., noise = 1e
    # Some problems are noisy but regardless is bad default behaviour!
    # DO NOT rebuild the model
    if not model_name in models:
      model = create_model(params)
      models[model_name]['history'] = run_model(models, params)
      summarise_history(models, model_names)

    print(model_name)
    bo_mse = models[model_name]['perf']['val_loss']['min']

    if reject_model(models[model_name]['perf'], strict = False):
      print("WARN: bad model", model_name)
      BAD_MODEL_PENALTY = 1
      bo_mse *= BAD_MODEL_PENALTY  # bad models get (arbitrarly) "higher" values

    return bo_mse


@use_named_args(dimensions = bo_dims_1s)
def model_fitness_1s(**dims):
    """This function is for illustrative purposes.
       The params values must be adapted for each optimisation task.
       Here default parameters for a single step-ahead stacked LSTM are used.
    """

    params = get_default_params('s_lstm', 1)

    return get_bo_mse(params, **dims)


def run_bo_search(bayes_opt, bo_id):

    # noise, limit but unfortunately not prevent re-evaluating the same point
    noise_level = 1e-10

    bo_search_results = gp_minimize(func = bayes_opt[bo_id]['fitness_func'],
                                    dimensions = bayes_opt[bo_id]['dims'],
                                    x0 = bayes_opt[bo_id]['init_dims'],
                                    n_calls = bayes_opt[bo_id]['calls'],
                                    acq_func = 'EI',
                                    noise = noise_level,
                                    verbose = True,
                                    random_state = 42)
```

```
    print()
    print(bo_search_results.x)
    print(bo_search_results.fun)
    print()

    plot_convergence(bo_search_results)

    plot_objective(result   = bo_search_results)
    plot_evaluations(result = bo_search_results)

    plot_bo_func_vals_dist(bo_search_results.func_vals, bo_id)

    return bo_search_results


def plot_bo_func_vals_dist(data, bo_results_id):
    """Plot skopt function values distribution using swarmplot and boxplot"""

    title = bo_results_id + ' gp_minimize function values - mse'

    fig1, ax1 = plt.subplots()
    ax1 = sns.swarmplot(y = data)
    ax1 = sns.boxplot(y = data,
                      showcaps = False,
                      boxprops = {'facecolor':'None', 'linewidth':1},
                      showfliers = False).set_title(title)
    plt.show()


hpo = {}  # hyperparameter optimisation
steps_str = '_48s'
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting scikit-optimize
  Downloading scikit_optimize-0.9.0-py2.py3-none-any.whl (100 kB)
     |████████████████████████████████| 100 kB 5.4 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
Collecting pyaml>=16.9
  Downloading pyaml-21.10.1-py2.py3-none-any.whl (24 kB)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-21.10.1 scikit-optimize-0.9.0

skopt version: 0.9.0
```

1. Simplified encoder decoder

I start with a simplified encoder decoder network built with the Keras sequential API. The encoder vector is taken from the final state of the encoder LSTM. Each time step of the LSTM outputs a hidden vector, but only the last one is used. The encoder vector is repeated n times, so each time step of the decoder LSTM receives the same vector. The RepeatVector layer is used to repeat the encoder vector. The decoder is also built with a LSTM layer but it outputs a vector at every time step. We then apply a Dense layer at every time step to predict one temperature at a time. The TimeDistributed layer applies the same Dense layer to every time step.

Note that, ususally the state from the encoder LSTM is used to initialise the decoder LSTM. Additionally, no teacher forcing is used.

Code for this architecture is in the `build_simple_encdec_model` function.

Briefly, the architecture is (omitting dropout and regularisation):

- encoder_state = LSTM(return_sequences = False)
- RepeatVector(encoder_state)
- LSTM(return_sequences = True)
- TimeDistributed(Dense(1))

Hardcoded parameters:

- batch_size 32

Optimise:

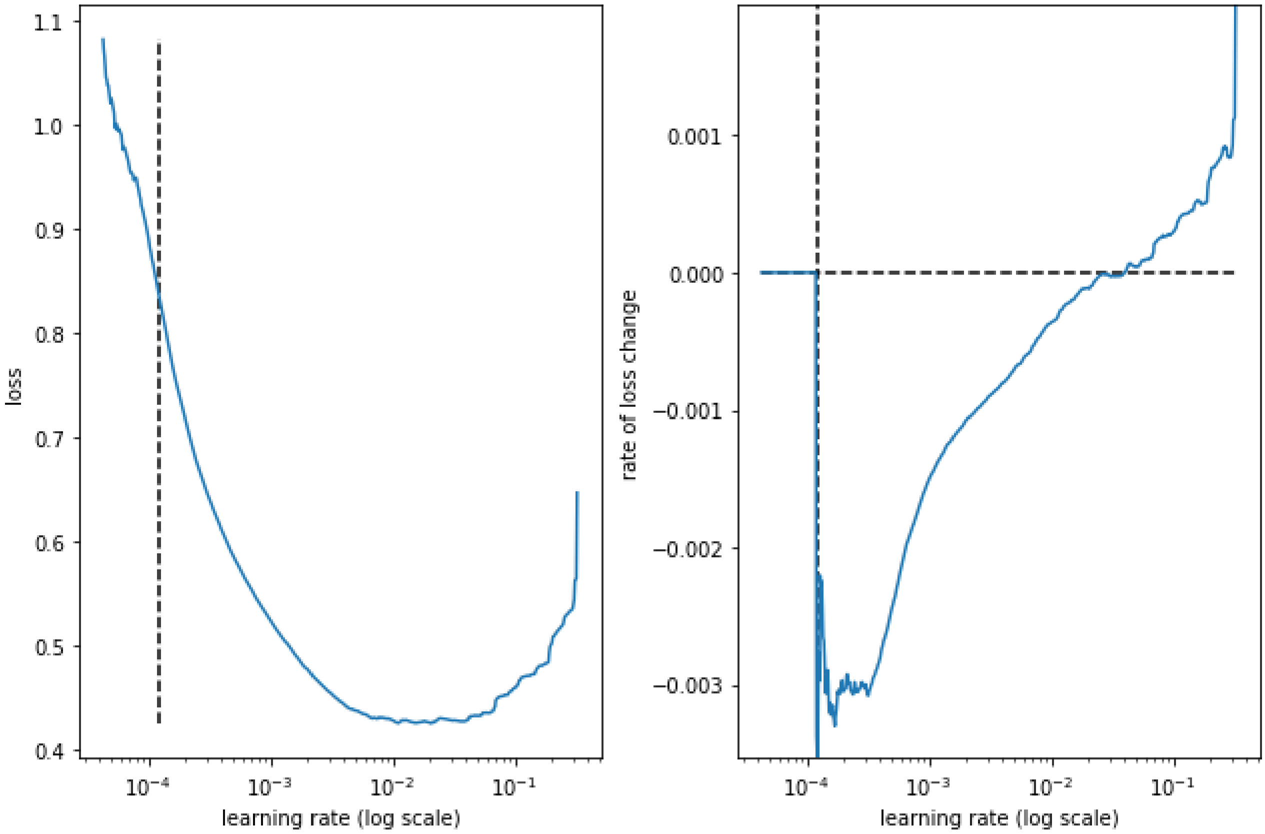
- lags
- feat_maps - LSTM feature maps

Unfortunately, Google Colab ran out of RAM during this optimisation run. I have reduced the skopt calls from 60 to 40.

```
%%time


mod_type   = 'simple'
results_id = mod_type + steps_str
hpo[results_id] = {}

dim_lags      = Integer(low = 24, high = 144, name = 'lags')
dim_feat_maps = Integer(low =  8, high =  64, name = 'feat_maps')

hpo[results_id]['dims'] = [dim_lags,
                           dim_feat_maps]
hpo[results_id]['init_dims'] = [24, 64]
hpo[results_id]['calls']     = 40


@use_named_args(dimensions = hpo[results_id]['dims'])
def model_fitness(**dims):
    params = get_default_params(mod_type)
    params.update({'lrf_params': [0.00003,  10, 32, 5, 100, 25]}) # better
```

```
    return get_bo_mse(params, **dims)


hpo[results_id]['fitness_func'] = model_fitness
hpo[results_id]['results']      = run_bo_search(hpo, results_id)


get_best_models(models)

display(rank_models(models, 'val_loss', strict = True, limit = 5))
display(rank_models(models, 'val_mae',  strict = True, limit = 5))
```

```
Iteration No: 1 started. Evaluating function at provided point.
lags 24
feat_maps 64
Epoch 1/5
5880/5880 [==============================] - 14s 1ms/step - loss: 11.3406 - ma
```




```
best lr: 0.00012141215

Model: "context_24l_48s_32bs_64fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 64)                19200

 repeat_vector (RepeatVector  (None, 48, 64)           0
 )

 lstm_1 (LSTM)               (None, 48, 64)            33024

 time_distributed (TimeDistr  (None, 48, 32)           2080
 ibuted)

 time_distributed_1 (TimeDis  (None, 48, 1)            33
 tributed)

=================================================================
Total params: 54,337
Trainable params: 54,337
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 50s - loss: 0.1527 - mae: 0.2960 - val_loss: 0.1354 - val_mae: 0.2
Epoch 2/5
5880/5880 - 43s - loss: 0.1123 - mae: 0.2549 - val_loss: 0.1296 - val_mae: 0.2
Epoch 3/5
```

Results for optimised simplified encoder decoder 48 steps-ahead forecast models after 5 epochs:

Encoder Decoder	params	mse	mae
Simplified	lags=49, feat_maps=54	0.12475	0.26074

Good model:

- context_49l_48s_32bs_54fm

Best model:

- context_142l_48s_32bs_36fm
- mse 0.120164
- mae 0.255025
- some signs of overfitting

The best of the simplified encoder decoder models shows some overfitting. The best simplified encoder decoder model without overfitting gives comparable performance to some of the best earlier LSTM and CNN models. Performance may be further improved with regularisation.

Gaussian process plots:

- lags - possibly periodic
  - primary minima around 24 - 48 (12 to 24 hours)
  - secondary minima around 144 (3 days)
  - small difference between primary and secondary minima
  - **surprising** maxima around 96 (2 days)
- feat_maps - again, primary and secondary minima
  - but possibly negligible difference

It's clear from some of the learning rate finder curves that start_lr and/or end_lr could be further refined. Start_lr may be a little low but results seem OK.

As with all of the skopt runs, it would benefit from running for more iterations and probably more learning rate tuning.

---

2. Autoencoder with attention

Autoencoders are typically used to learn a representation (an encoding) for a data set by attempting to regenerate the input from the representation. An encoder model learns the representation and a separate decoder model regenerates the input from the representation.

Alternatively, the decoder can learn a target sequence and we can use "attention" to both align and translate sequences. Alignment identifies which parts of the input sequence are relevant to each part in the output sequence. Translation is the process of using the relevant information to select the appropriate output.

Attention

If you squint from 40,000 feet, then attention looks like a glorified weighting scheme. It boosts some parts of the data and diminishes other parts. Gradient descent is used to learn which parts of the data to apply more or less attention to.

The attention mechanism was popularised by the Attention Is All You Need paper, but first appeared in Neural Machine Translation by Jointly Learning to Align and Translate.

Code for this architecture is in the `build_autoencoder_model` function.

The autoencoder architecture is composed of 3 models. Briefly, the architecture is (omitting dropout and regularisation):

1. encoder model:
    - LSTM()
    - Attention() - Luong-style multiplicative scoring function
2. decoder model:
    - LSTM()
    - AdditiveAttention() - Bahdanau-style additive scoring function
    - GlobalAveragePooling1D()
    - Dense()
3. autoencoder model:
    - encoder + decoder
        - input data (X): past observations
        - output data (y): future temperatures to forecast

What is the difference between Luong attention and Bahdanau attention?

Teacher forcing is not used and there is no need for a custom predict sequence function.

Hardcoded parameters:

- batch_size 32

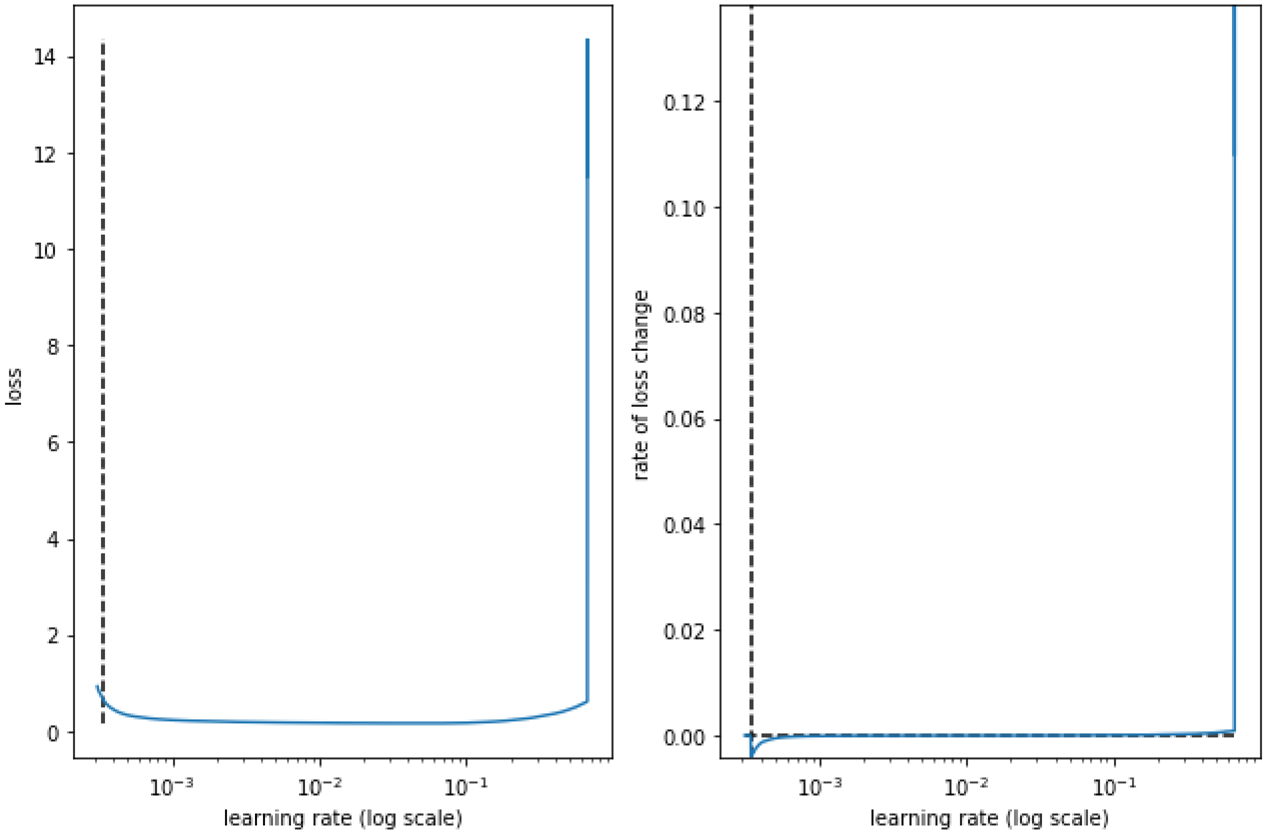
Optimise:

- lags
- feat_maps - LSTM feature maps

```
%%time


mod_type   = 'auto'
results_id = mod_type + steps_str
hpo[results_id] = {}

dim_lags       = Integer(low = 24, high = 144, name = 'lags')
```

```
dim_feat_maps = Integer(low =  8, high =  64, name = 'feat_maps')

hpo[results_id]['dims'] = [dim_lags,
                           dim_feat_maps]
hpo[results_id]['init_dims'] = [24, 64]
hpo[results_id]['calls']     = 40


@use_named_args(dimensions = hpo[results_id]['dims'])
def model_fitness(**dims):
    params = get_default_params(mod_type)
    params.update({'epochs':  5,
                   'lrf_params': [0.0003,  0.001, 32, 5, 100, 25]})
  # params.update({'lrf_params': [0.003, 10, 32, 5, 100, 25]})
  # params.update({'lrf_params': [0.000003, 10, 32, 5, 100, 25]})
  # params.update({'lrf_params': [0.00003,  10, 32, 5, 100, 25]})  # better??
  # params.update({'lrf_params': [0.0003,   10, 32, 5, 100, 25]})

    return get_bo_mse(params, **dims)


hpo[results_id]['fitness_func'] = model_fitness
hpo[results_id]['results']      = run_bo_search(hpo, results_id)


get_best_models(models)

display(rank_models(models, 'val_loss', strict = True, limit = 5))
display(rank_models(models, 'val_mae',  strict = True, limit = 5))
```

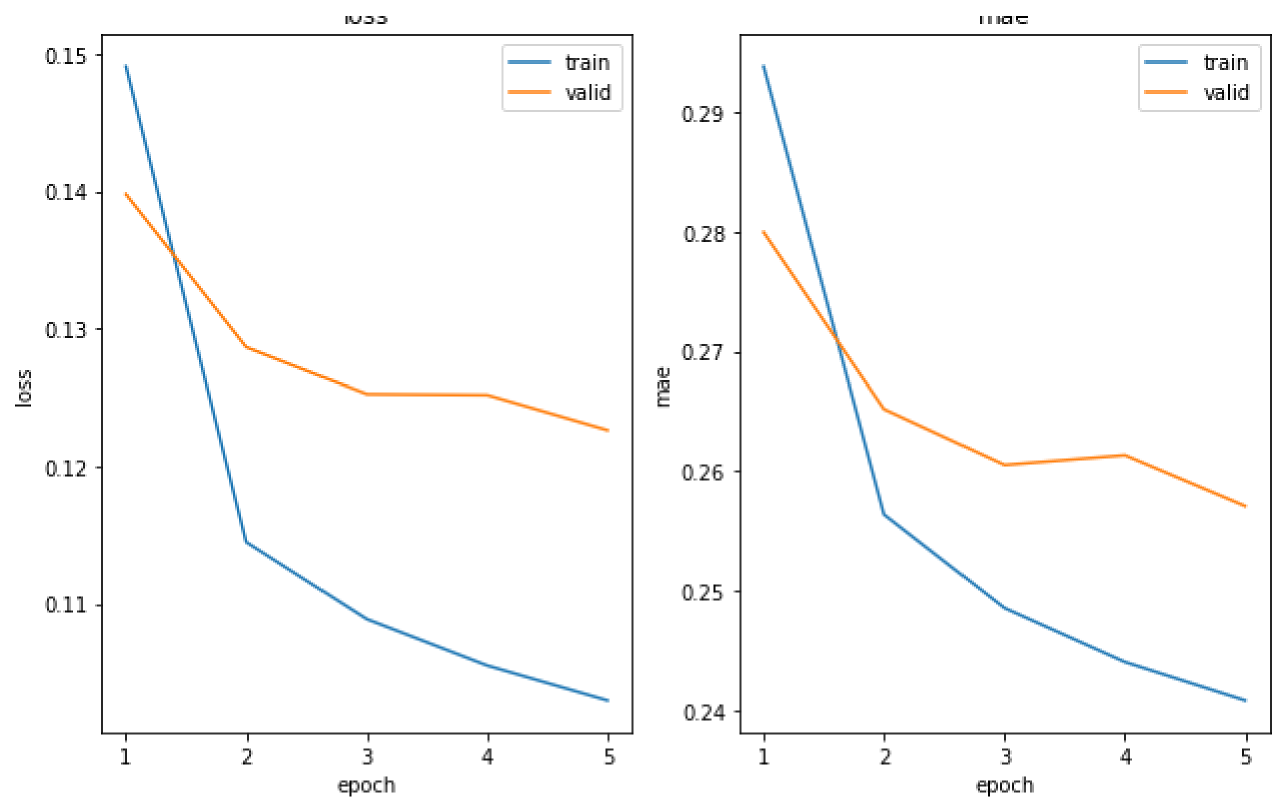
```
Iteration No: 1 started. Evaluating function at provided point.
lags 24
feat_maps 64
Epoch 1/5
5880/5880 [==============================] - 52s 8ms/step - loss: 0.6182 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 8us/step - loss: 6.6003 - mae:
```



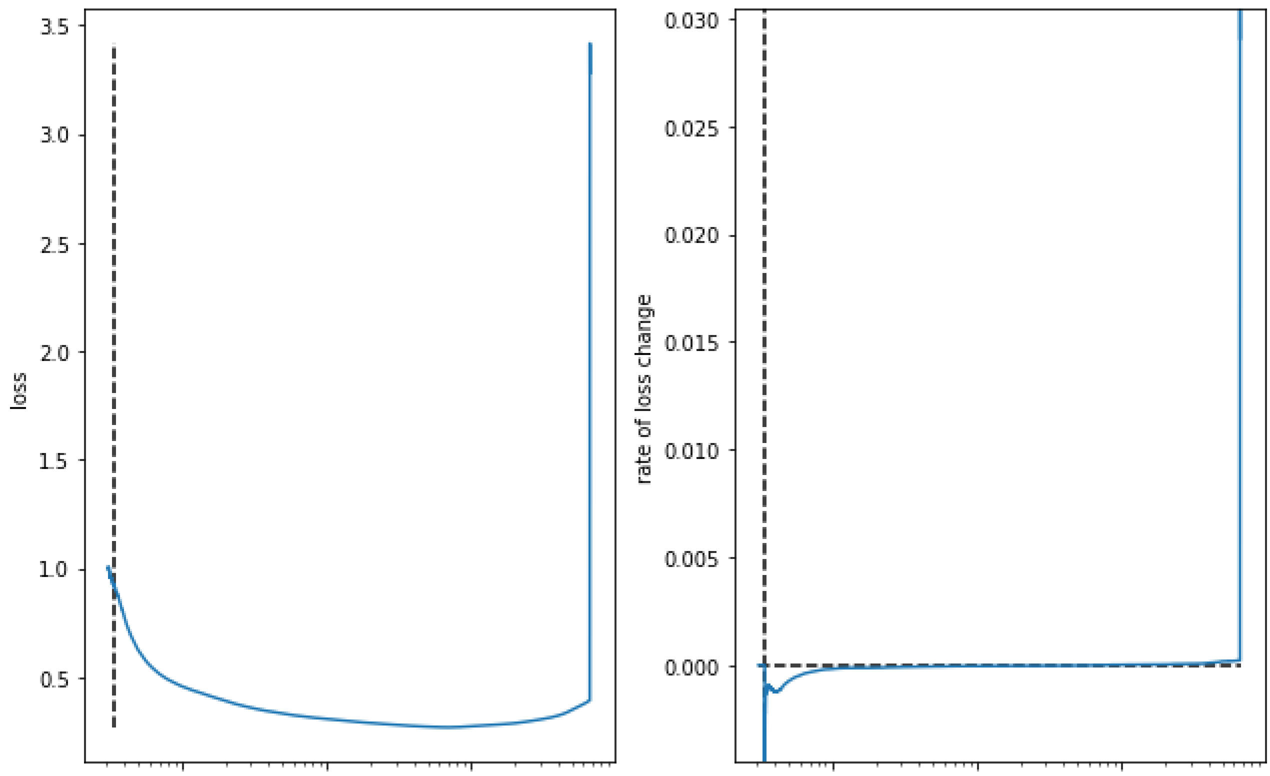

```
best lr: 0.0003424541

Model: "auto_24l_48s_32bs_64fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 24, 10)]          0

 encoder (Functional)        (None, 24, 64)            19200

 decoder (Functional)        (None, 48)                36208

=================================================================
Total params: 55,408
Trainable params: 55,408
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 38s - loss: 0.1491 - mae: 0.2938 - val_loss: 0.1398 - val_mae: 0.2
Epoch 2/5
5880/5880 - 37s - loss: 0.1145 - mae: 0.2564 - val_loss: 0.1287 - val_mae: 0.2
Epoch 3/5
5880/5880 - 35s - loss: 0.1089 - mae: 0.2486 - val_loss: 0.1253 - val_mae: 0.2
Epoch 4/5
5880/5880 - 34s - loss: 0.1055 - mae: 0.2440 - val_loss: 0.1252 - val_mae: 0.2
Epoch 5/5
5880/5880 - 35s - loss: 0.1030 - mae: 0.2408 - val_loss: 0.1226 - val_mae: 0.2
```

auto_24l_48s_32bs_64fm loss

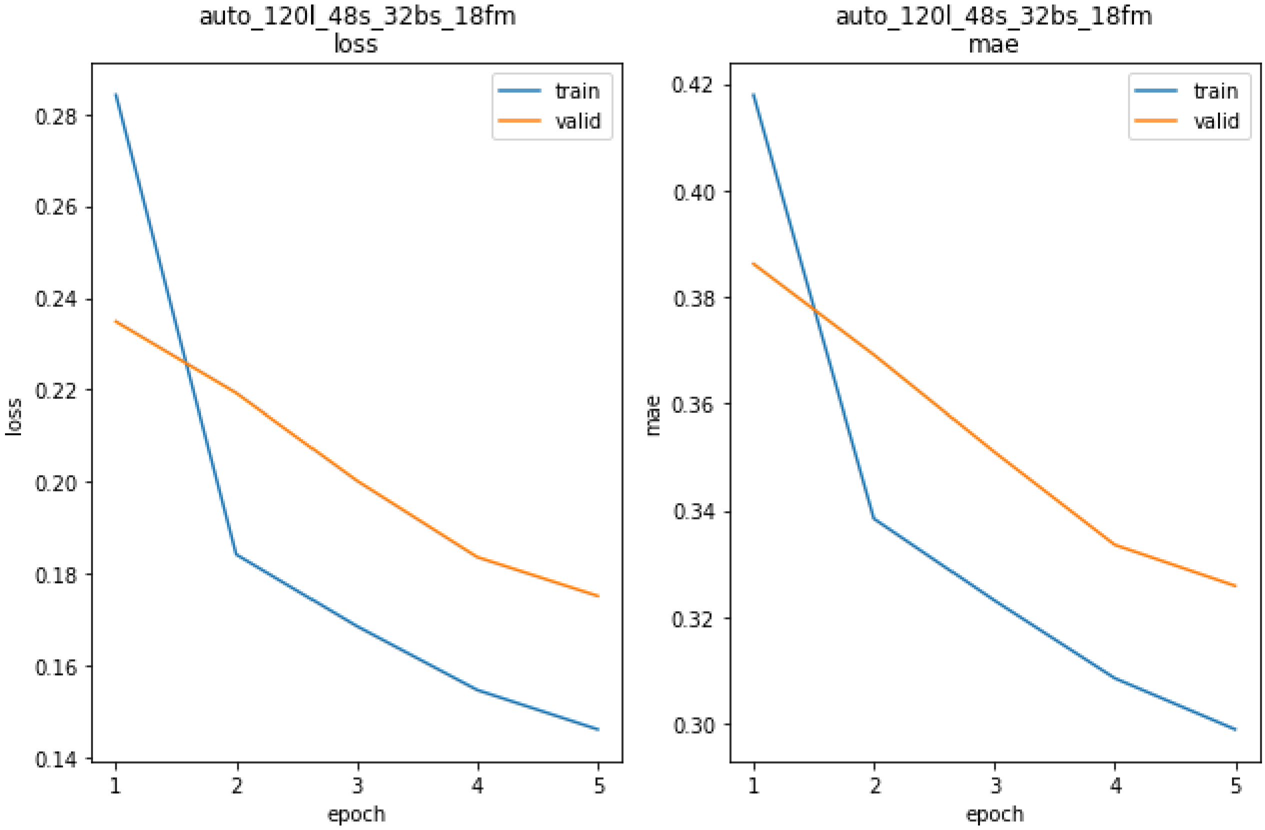
auto_24l_48s_32bs_64fm mae




```
auto_24l_48s_32bs_64fm train min loss: 0.102959 mae: 0.240832   epoch: 5
auto_24l_48s_32bs_64fm valid min loss: 0.122634 mae: 0.257099   epoch: 5

auto_24l_48s_32bs_64fm
Iteration No: 1 ended. Evaluation done at provided point.
Time taken: 240.4803
Function value obtained: 0.1226
Current minimum: 0.1226
Iteration No: 2 started. Evaluating function at random point.
lags 120
feat_maps 18
Epoch 1/5
5877/5877 [==============================] - 94s 16ms/step - loss: 0.3938 - ma
Epoch 2/5
5877/5877 [==============================] - 0s 16us/step - loss: 2.5581 - mae
```

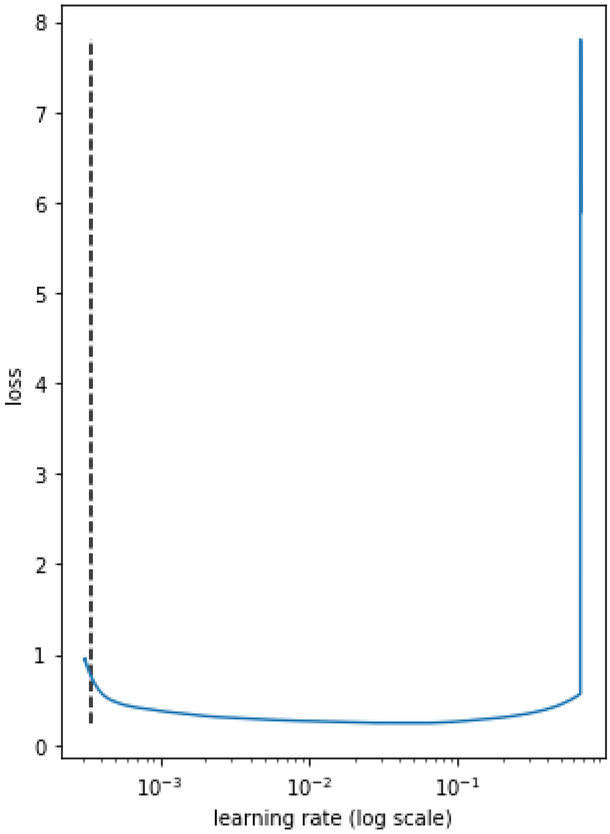


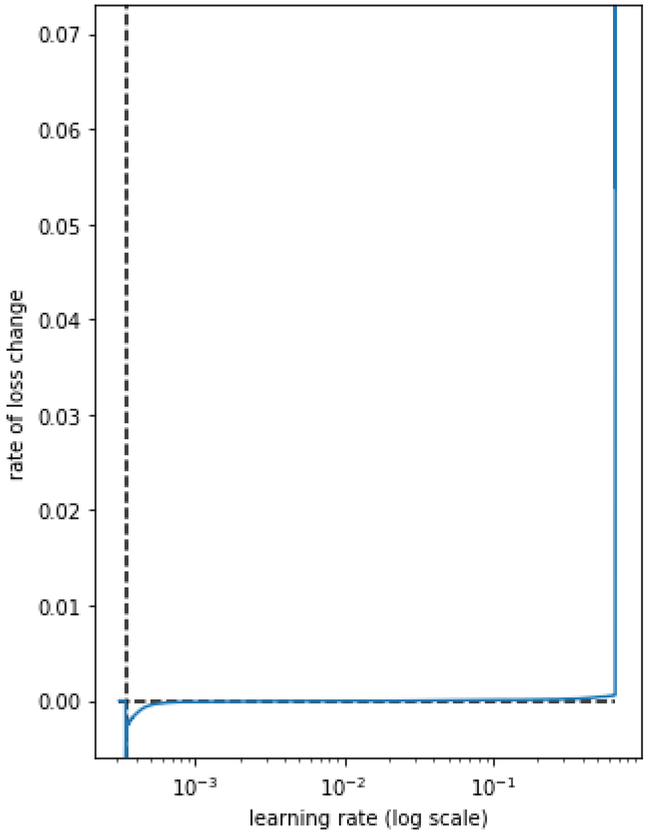

```
best lr: 0.00034202848

Model: "auto_120l_48s_32bs_18fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 120, 10)]         0         
                                                                 
 encoder (Functional)        (None, 120, 18)           2088      
                                                                 
 decoder (Functional)        (None, 48)                3594      
                                                                 
=================================================================
Total params: 5,682
Trainable params: 5,682
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5877/5877 - 86s - loss: 0.2841 - mae: 0.4179 - val_loss: 0.2348 - val_mae: 0.3
Epoch 2/5
5877/5877 - 83s - loss: 0.1842 - mae: 0.3384 - val_loss: 0.2193 - val_mae: 0.3
Epoch 3/5
5877/5877 - 83s - loss: 0.1686 - mae: 0.3231 - val_loss: 0.2002 - val_mae: 0.3
Epoch 4/5
5877/5877 - 83s - loss: 0.1547 - mae: 0.3085 - val_loss: 0.1836 - val_mae: 0.3
Epoch 5/5
5877/5877 - 83s - loss: 0.1461 - mae: 0.2989 - val_loss: 0.1752 - val_mae: 0.3
```




```
auto_120l_48s_32bs_18fm train min loss: 0.146116         mae: 0.298885    epoch:
auto_120l_48s_32bs_18fm valid min loss: 0.175162         mae: 0.325823    epoch:

auto_120l_48s_32bs_18fm
Iteration No: 2 ended. Evaluation done at random point.
Time taken: 573.6857
Function value obtained: 0.1752
```

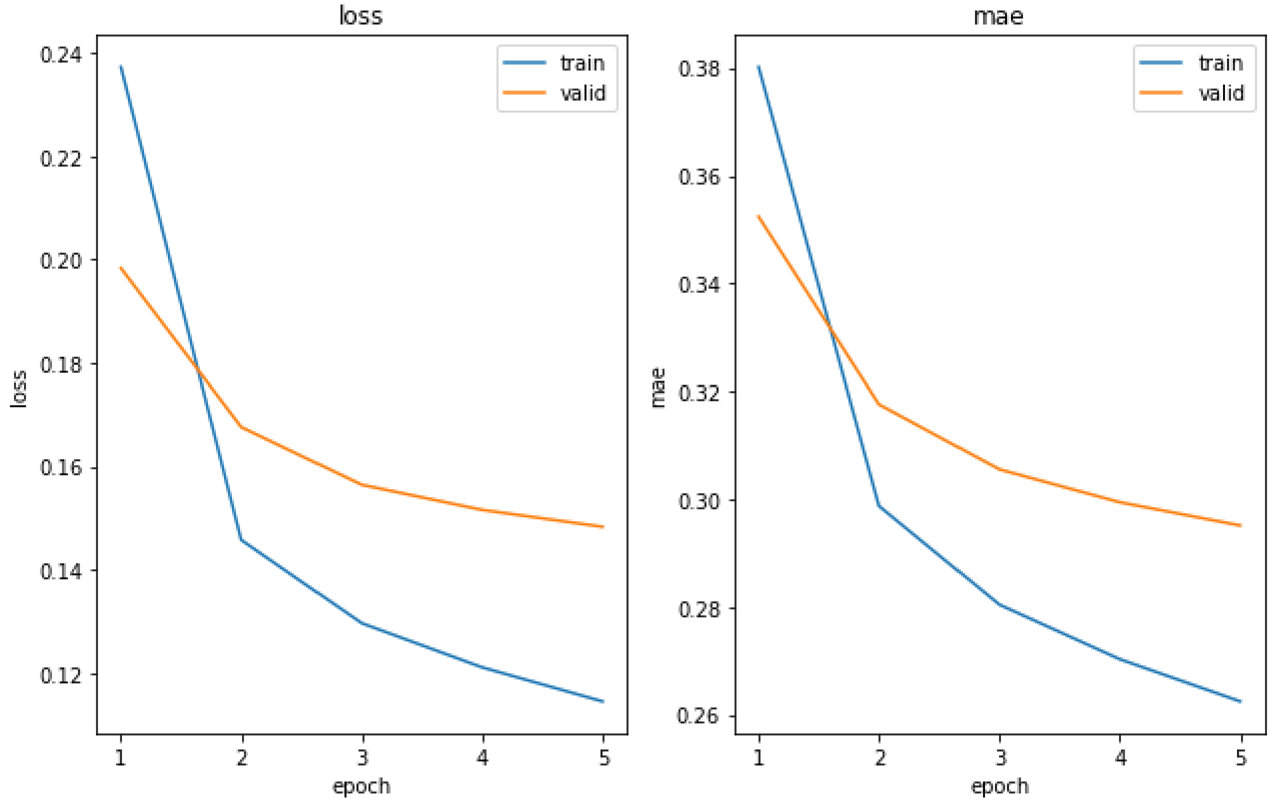
```
Current minimum: 0.1226
Iteration No: 3 started. Evaluating function at random point.
lags 118
feat_maps 41
Epoch 1/5
5877/5877 [==============================] - 114s 19ms/step - loss: 0.5649 - m
Epoch 2/5
5877/5877 [==============================] - 0s 19us/step - loss: 4.1004 - mae
```



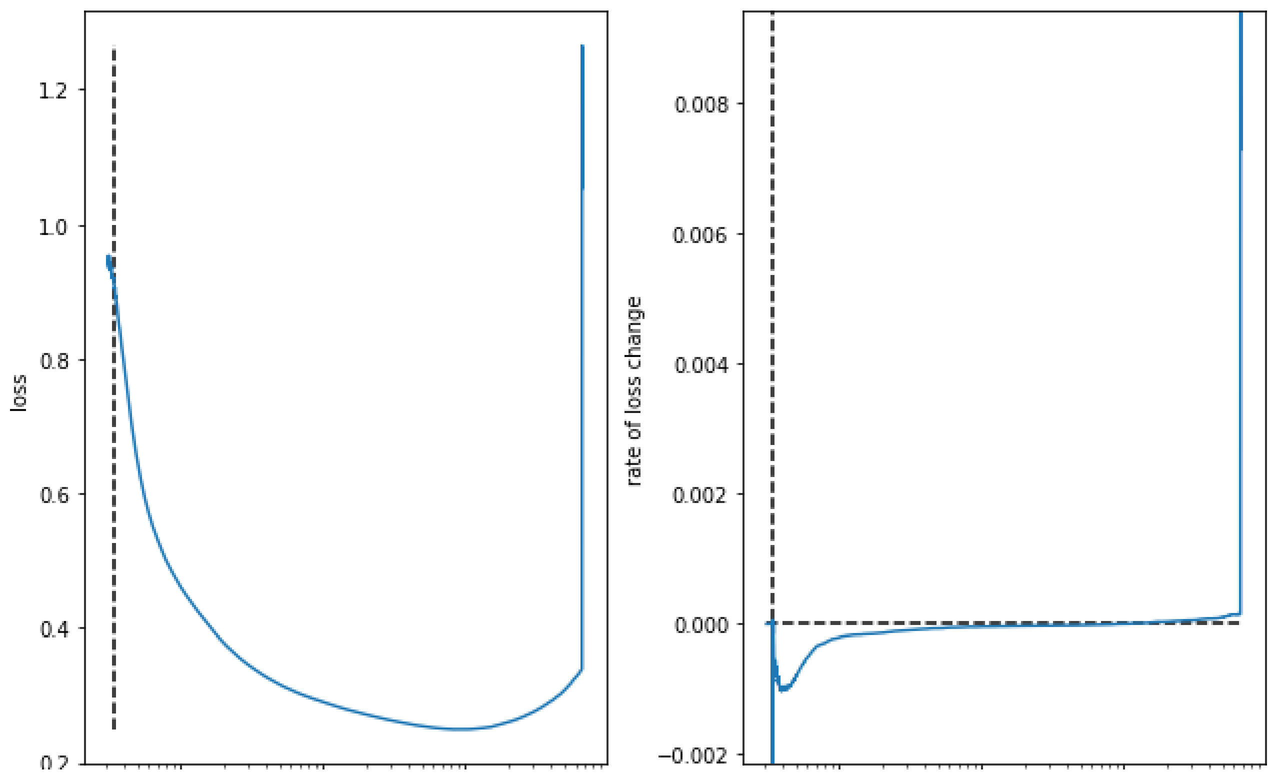



```
best lr: 0.00034202848

Model: "auto_118l_48s_32bs_41fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 118, 10)]         0

 encoder (Functional)        (None, 118, 41)           8528

 decoder (Functional)        (None, 48)                15669

=================================================================
Total params: 24,197
Trainable params: 24,197
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5877/5877 - 105s - loss: 0.2373 - mae: 0.3802 - val_loss: 0.1983 - val_mae: 0.
Epoch 2/5
5877/5877 - 102s - loss: 0.1458 - mae: 0.2988 - val_loss: 0.1676 - val_mae: 0.
Epoch 3/5
5877/5877 - 102s - loss: 0.1296 - mae: 0.2804 - val_loss: 0.1564 - val_mae: 0.
Epoch 4/5
5877/5877 - 102s - loss: 0.1211 - mae: 0.2703 - val_loss: 0.1516 - val_mae: 0.
Epoch 5/5
5877/5877 - 103s - loss: 0.1145 - mae: 0.2625 - val_loss: 0.1483 - val_mae: 0.
```

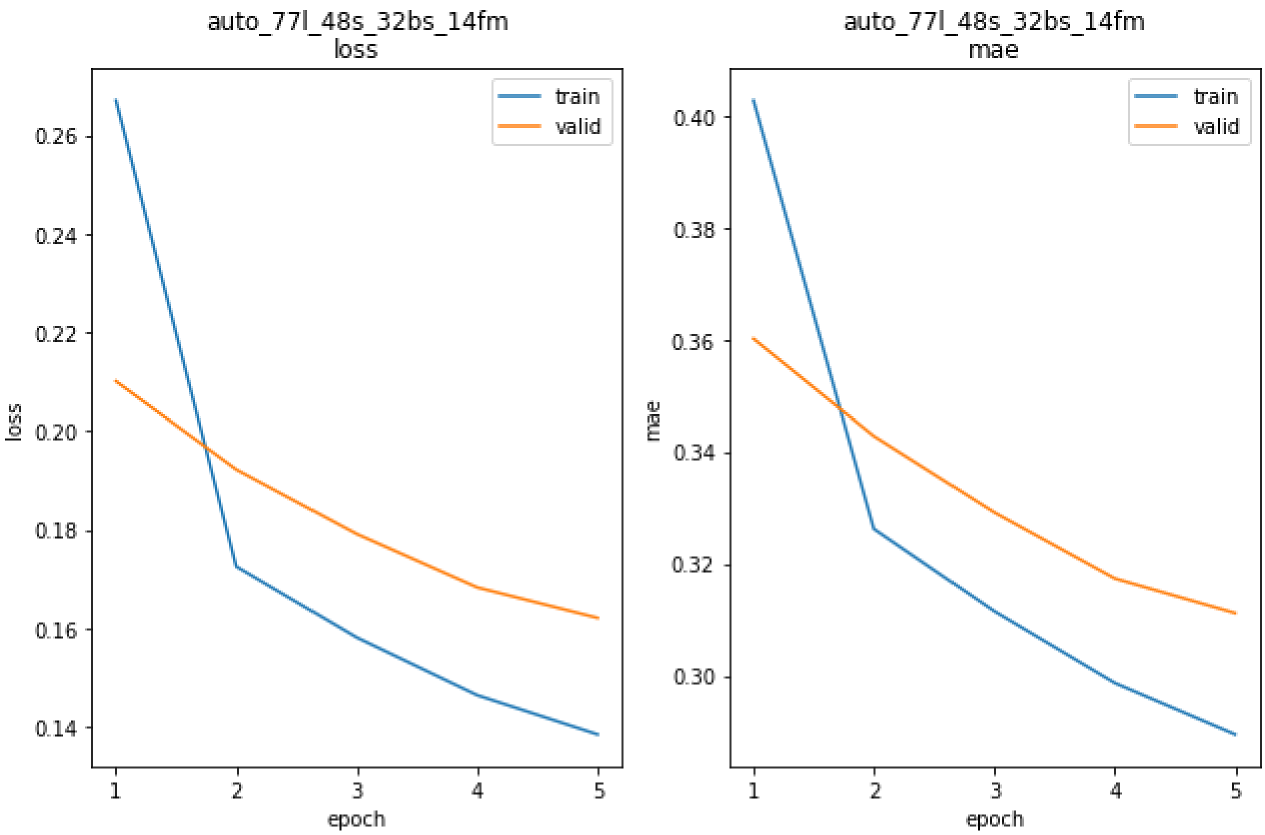
auto_118l_48s_32bs_41fm auto_118l_48s_32bs_41fm




```
auto_118l_48s_32bs_41fm train min loss: 0.114495        mae: 0.262496    epoch:
auto_118l_48s_32bs_41fm valid min loss: 0.148312        mae: 0.295153    epoch:

auto_118l_48s_32bs_41fm
Iteration No: 3 ended. Evaluation done at random point.
Time taken: 670.2167
Function value obtained: 0.1483
Current minimum: 0.1226
Iteration No: 4 started. Evaluating function at random point.
lags 77
feat_maps 14
Epoch 1/5
5878/5878 [==============================] - 64s 10ms/step - loss: 0.3392 - ma
Epoch 2/5
5878/5878 [==============================] - 0s 10us/step - loss: 1.0746 - mae
```

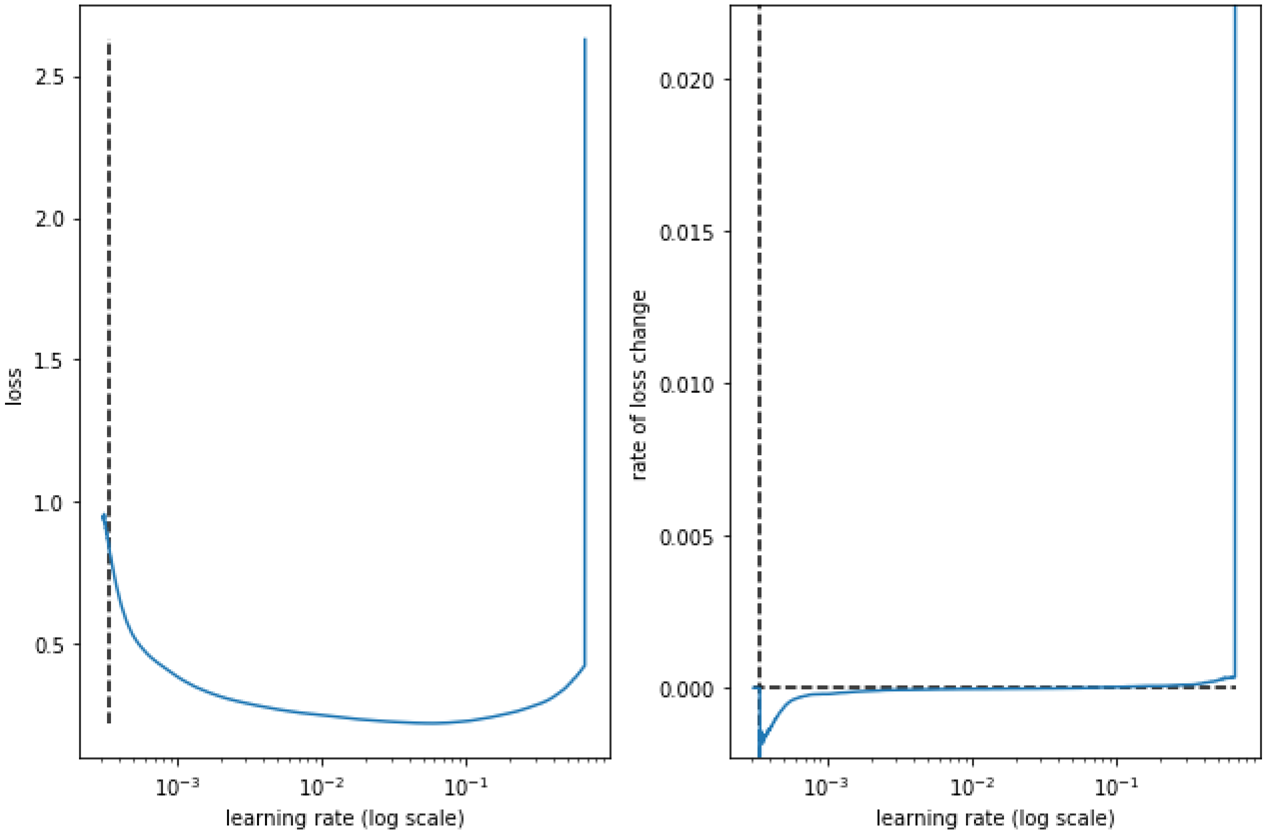



```
best lr: 0.00034202088

Model: "auto_77l_48s_32bs_14fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 77, 10)]          0

 encoder (Functional)        (None, 77, 14)            1400

 decoder (Functional)        (None, 48)                2358

=================================================================
Total params: 3,758
Trainable params: 3,758
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5878/5878 - 56s - loss: 0.2671 - mae: 0.4027 - val_loss: 0.2101 - val_mae: 0.3
Epoch 2/5
5878/5878 - 53s - loss: 0.1725 - mae: 0.3262 - val_loss: 0.1922 - val_mae: 0.3
Epoch 3/5
5878/5878 - 53s - loss: 0.1581 - mae: 0.3115 - val_loss: 0.1791 - val_mae: 0.3
Epoch 4/5
5878/5878 - 53s - loss: 0.1464 - mae: 0.2987 - val_loss: 0.1683 - val_mae: 0.3
Epoch 5/5
5878/5878 - 53s - loss: 0.1384 - mae: 0.2895 - val_loss: 0.1621 - val_mae: 0.3
```

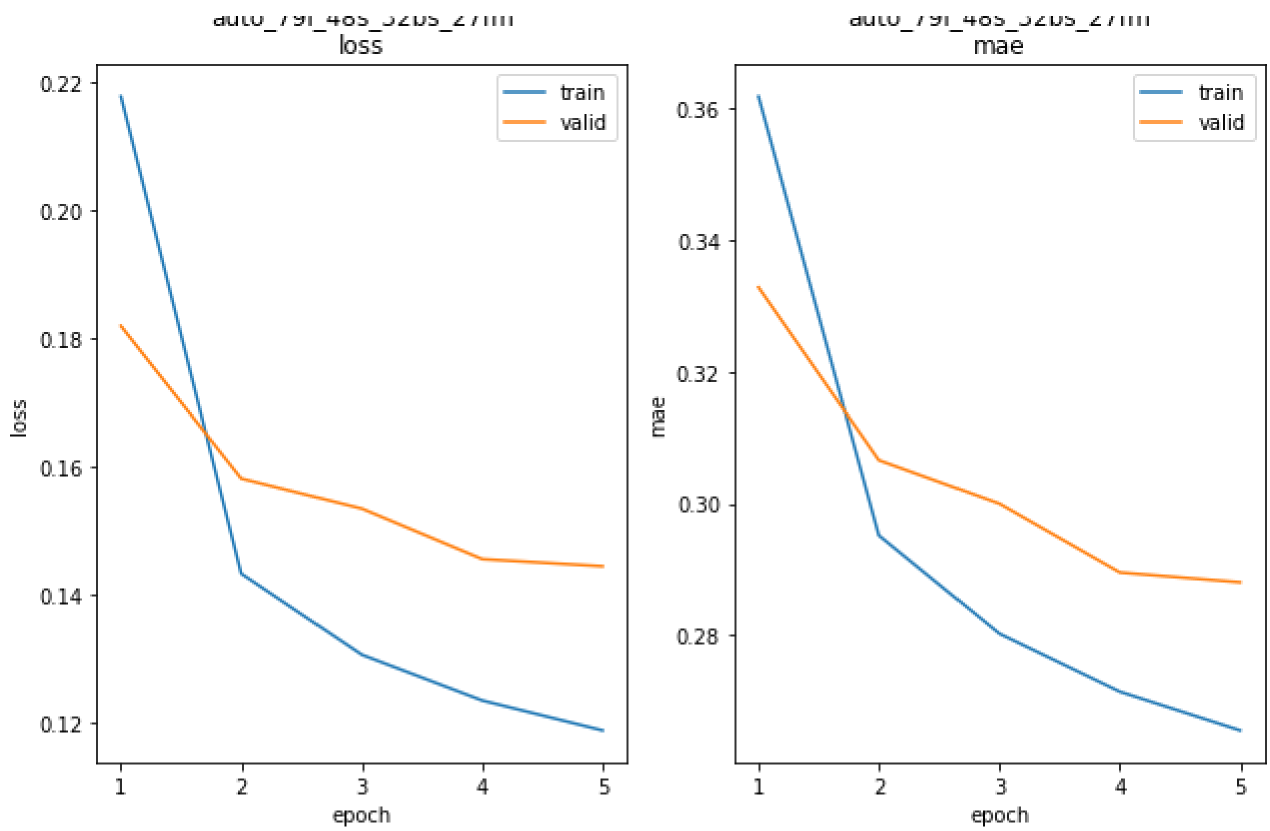



```
auto_77l_48s_32bs_14fm train min loss: 0.138432 mae: 0.289488   epoch: 5
auto_77l_48s_32bs_14fm valid min loss: 0.162092 mae: 0.311173   epoch: 5

auto_77l_48s_32bs_14fm
Iteration No: 4 ended. Evaluation done at random point.
Time taken: 335.3532
Function value obtained: 0.1621
```

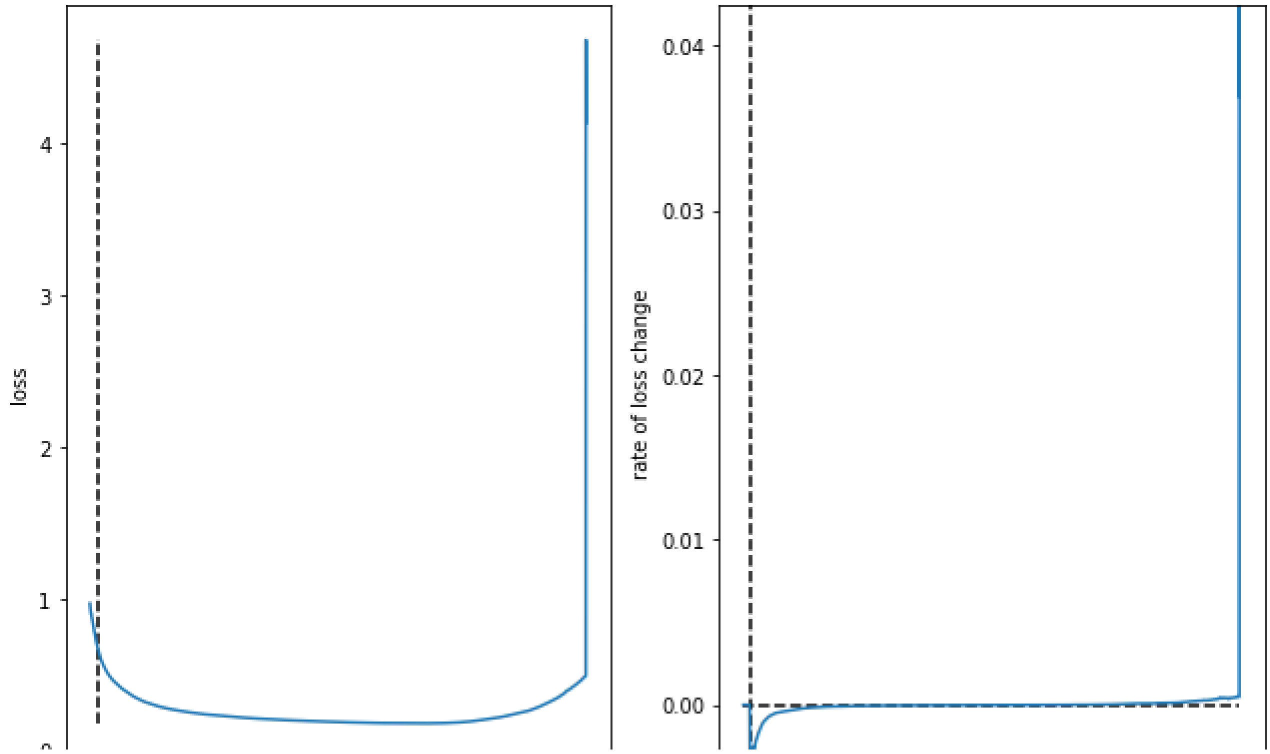
```
Current minimum: 0.1226
Iteration No: 5 started. Evaluating function at random point.
lags 79
feat_maps 27
Epoch 1/5
5878/5878 [==============================] - 73s 12ms/step - loss: 0.4242 - ma
Epoch 2/5
5878/5878 [==============================] - 0s 12us/step - loss: 3.3660 - mae
```

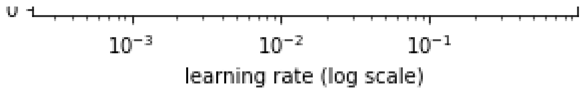


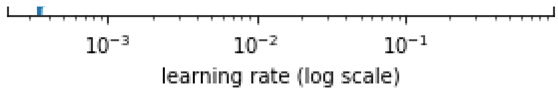

```
best lr: 0.0003433686

Model: "auto_79l_48s_32bs_27fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 79, 10)]          0

 encoder (Functional)        (None, 79, 27)            4104

 decoder (Functional)        (None, 48)                7311

=================================================================
Total params: 11,415
Trainable params: 11,415
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5878/5878 - 64s - loss: 0.2177 - mae: 0.3619 - val_loss: 0.1818 - val_mae: 0.3
Epoch 2/5
5878/5878 - 60s - loss: 0.1432 - mae: 0.2951 - val_loss: 0.1581 - val_mae: 0.3
Epoch 3/5
5878/5878 - 60s - loss: 0.1305 - mae: 0.2802 - val_loss: 0.1534 - val_mae: 0.3
Epoch 4/5
5878/5878 - 60s - loss: 0.1234 - mae: 0.2713 - val_loss: 0.1455 - val_mae: 0.2
Epoch 5/5
5878/5878 - 60s - loss: 0.1187 - mae: 0.2655 - val_loss: 0.1444 - val_mae: 0.2
```



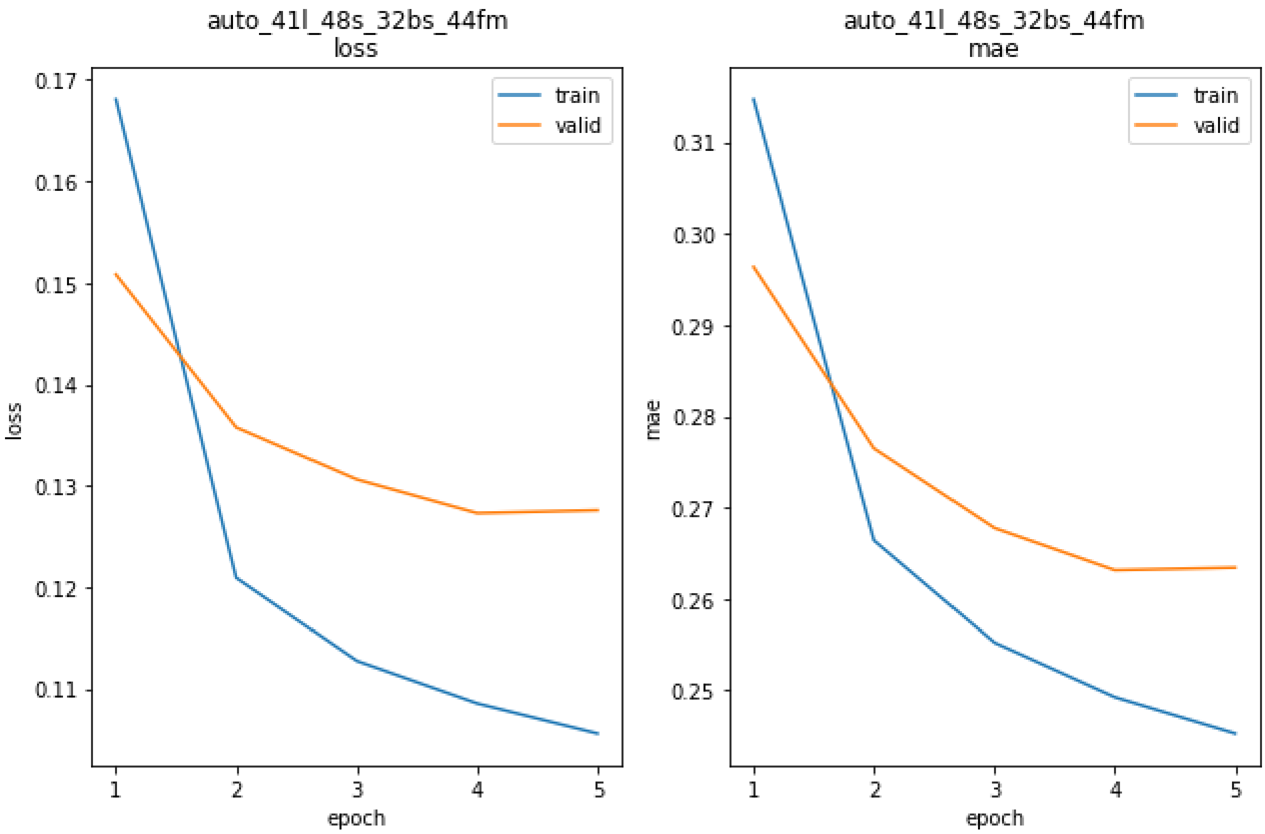

```
auto_79l_48s_32bs_27fm train min loss: 0.118726 mae: 0.265451   epoch: 5
auto_79l_48s_32bs_27fm valid min loss: 0.144372 mae: 0.287998   epoch: 5

auto_79l_48s_32bs_27fm
Iteration No: 5 ended. Evaluation done at random point.
Time taken: 379.9607
Function value obtained: 0.1444
Current minimum: 0.1226
Iteration No: 6 started. Evaluating function at random point.
lags 41
feat_maps 44
Epoch 1/5
5879/5879 [==============================] - 53s 8ms/step - loss: 0.4943 - mae
Epoch 2/5
5879/5879 [==============================] - 0s 10us/step - loss: 3.4874 - mae
```





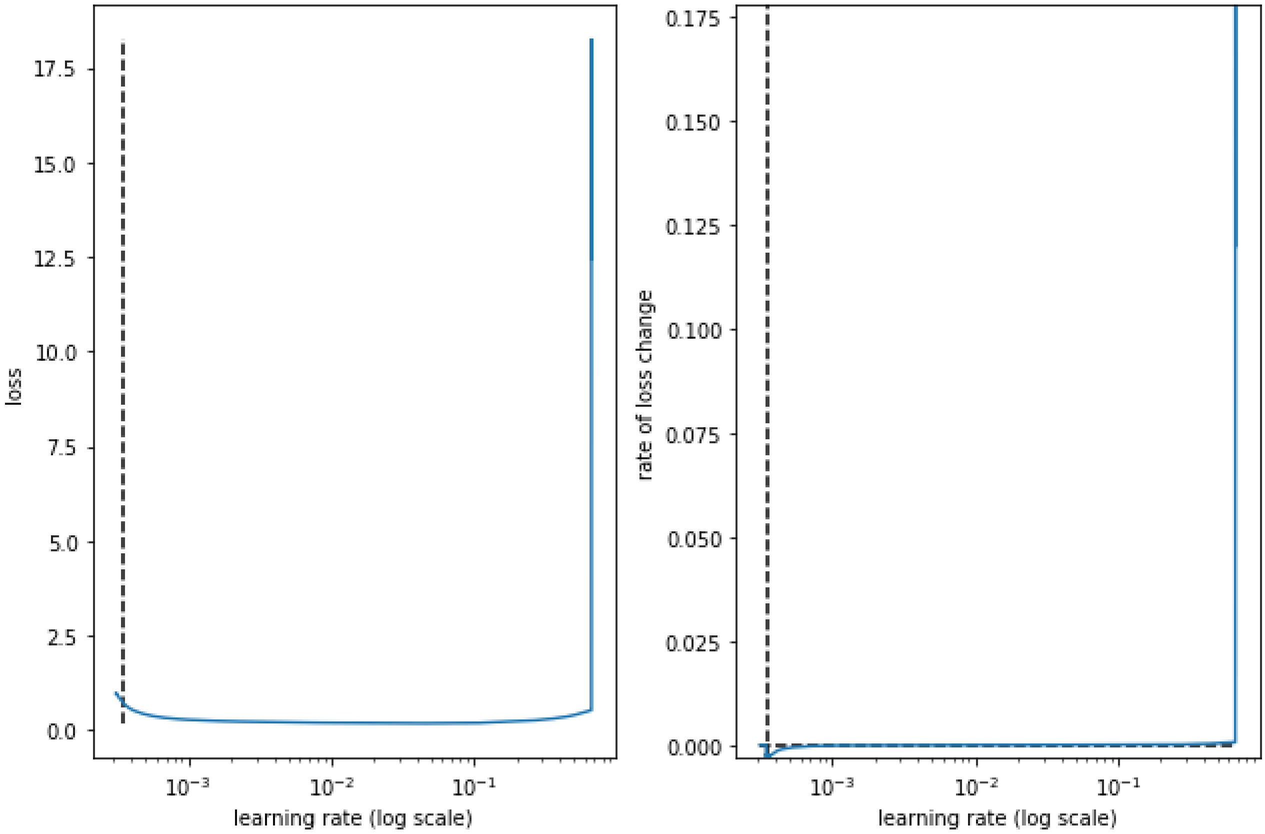



```
best lr: 0.00035109717

Model: "auto_41l_48s_32bs_44fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 41, 10)]          0

 encoder (Functional)        (None, 41, 44)            9680

 decoder (Functional)        (None, 48)                17868

=================================================================
Total params: 27,548
Trainable params: 27,548
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5879/5879 - 44s - loss: 0.1681 - mae: 0.3147 - val_loss: 0.1508 - val_mae: 0.2
Epoch 2/5
5879/5879 - 40s - loss: 0.1209 - mae: 0.2664 - val_loss: 0.1358 - val_mae: 0.2
Epoch 3/5
5879/5879 - 40s - loss: 0.1127 - mae: 0.2552 - val_loss: 0.1306 - val_mae: 0.2
Epoch 4/5
5879/5879 - 40s - loss: 0.1085 - mae: 0.2493 - val_loss: 0.1273 - val_mae: 0.2
Epoch 5/5
5879/5879 - 40s - loss: 0.1056 - mae: 0.2453 - val_loss: 0.1276 - val_mae: 0.2
```

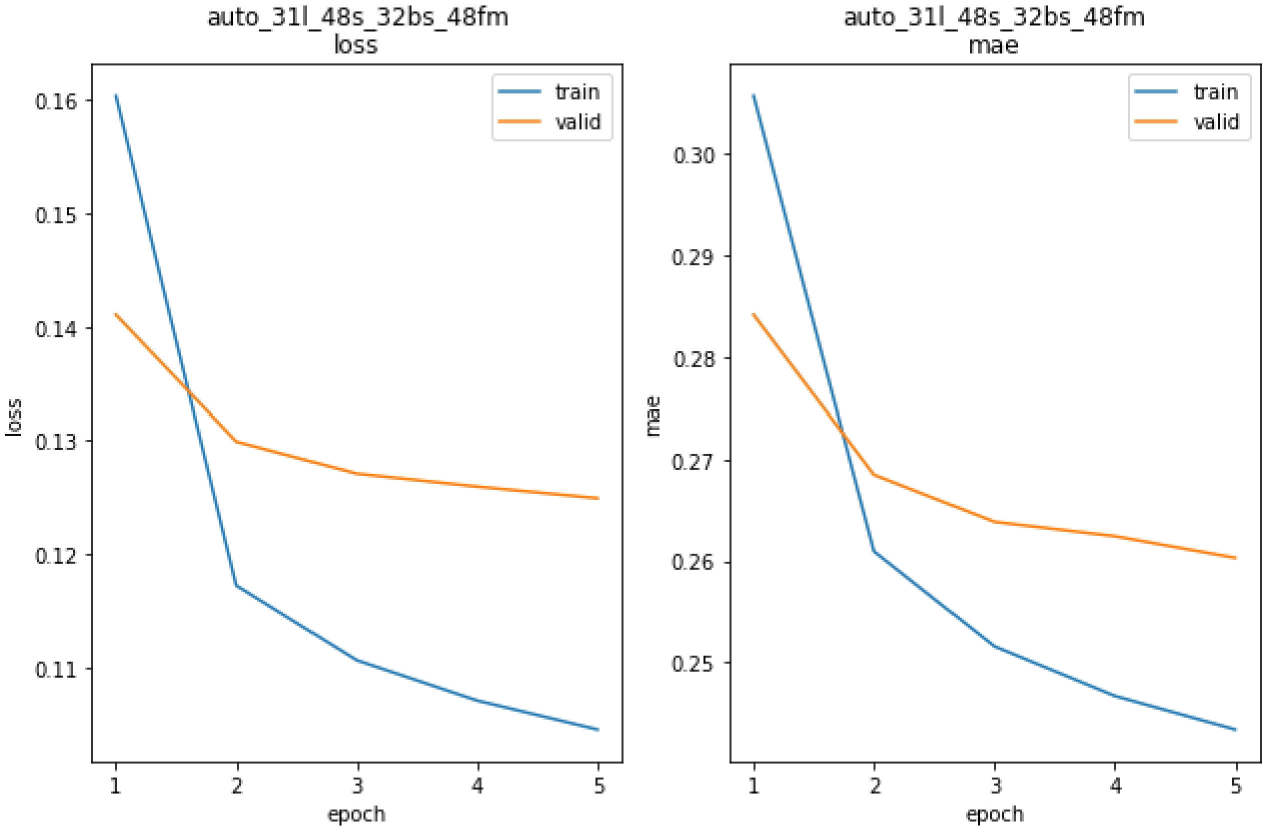



```
auto_41l_48s_32bs_44fm train min loss: 0.105588 mae: 0.245296   epoch: 5
auto_41l_48s_32bs_44fm valid min loss: 0.127290 mae: 0.263151   epoch: 4

auto_41l_48s_32bs_44fm
Iteration No: 6 ended. Evaluation done at random point.
Time taken: 259.6430
```

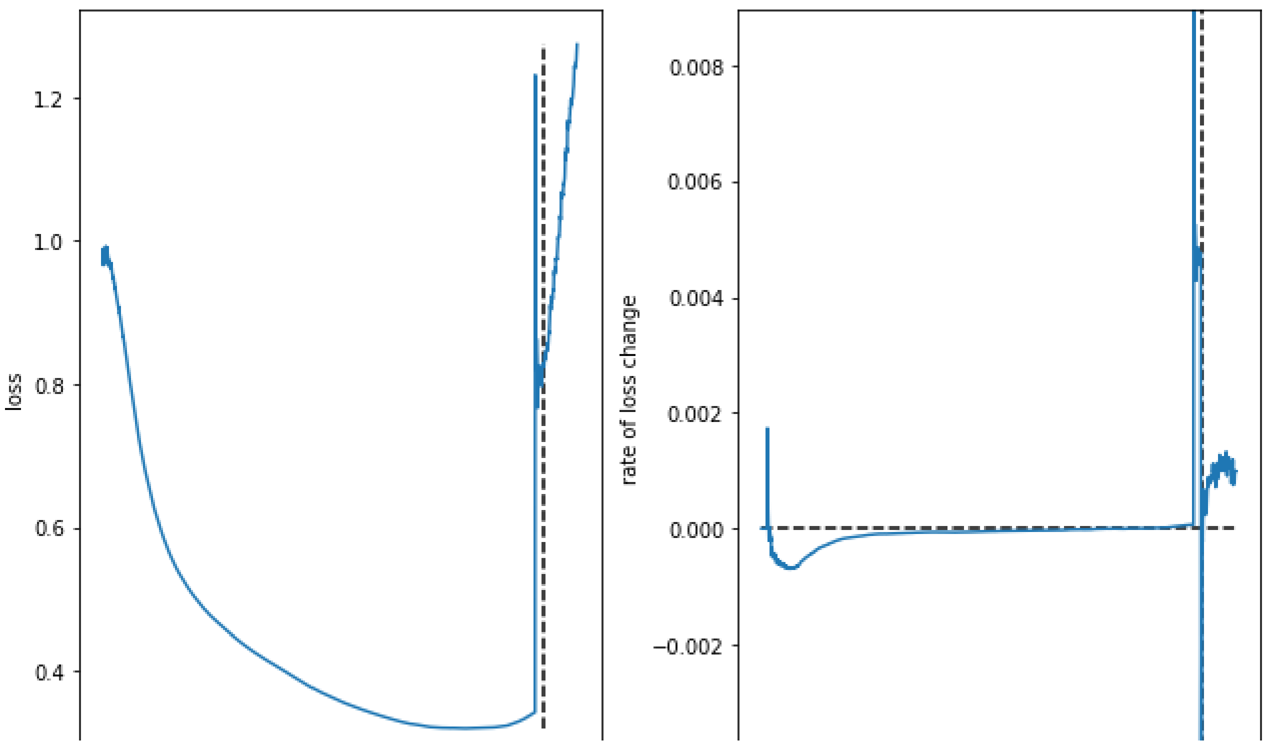
```
Function value obtained: 0.1273
Current minimum: 0.1226
Iteration No: 7 started. Evaluating function at random point.
lags 31
feat_maps 48
Epoch 1/5
5880/5880 [==============================] - 51s 8ms/step - loss: 0.5241 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 8us/step - loss: 5.4607 - mae:
```

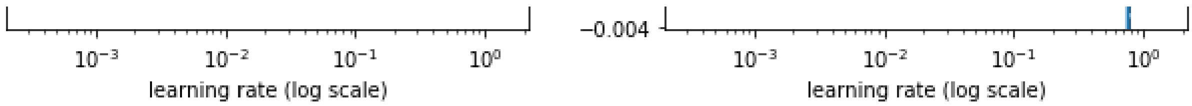



```
best lr: 0.00034833804

Model: "auto_31l_48s_32bs_48fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 31, 10)]          0

 encoder (Functional)        (None, 31, 48)            11328

 decoder (Functional)        (None, 48)                21024

=================================================================
Total params: 32,352
Trainable params: 32,352
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 39s - loss: 0.1604 - mae: 0.3057 - val_loss: 0.1411 - val_mae: 0.2
Epoch 2/5
5880/5880 - 36s - loss: 0.1172 - mae: 0.2609 - val_loss: 0.1299 - val_mae: 0.2
Epoch 3/5
5880/5880 - 36s - loss: 0.1106 - mae: 0.2516 - val_loss: 0.1271 - val_mae: 0.2
Epoch 4/5
5880/5880 - 36s - loss: 0.1070 - mae: 0.2467 - val_loss: 0.1259 - val_mae: 0.2
Epoch 5/5
5880/5880 - 36s - loss: 0.1045 - mae: 0.2434 - val_loss: 0.1249 - val_mae: 0.2
```

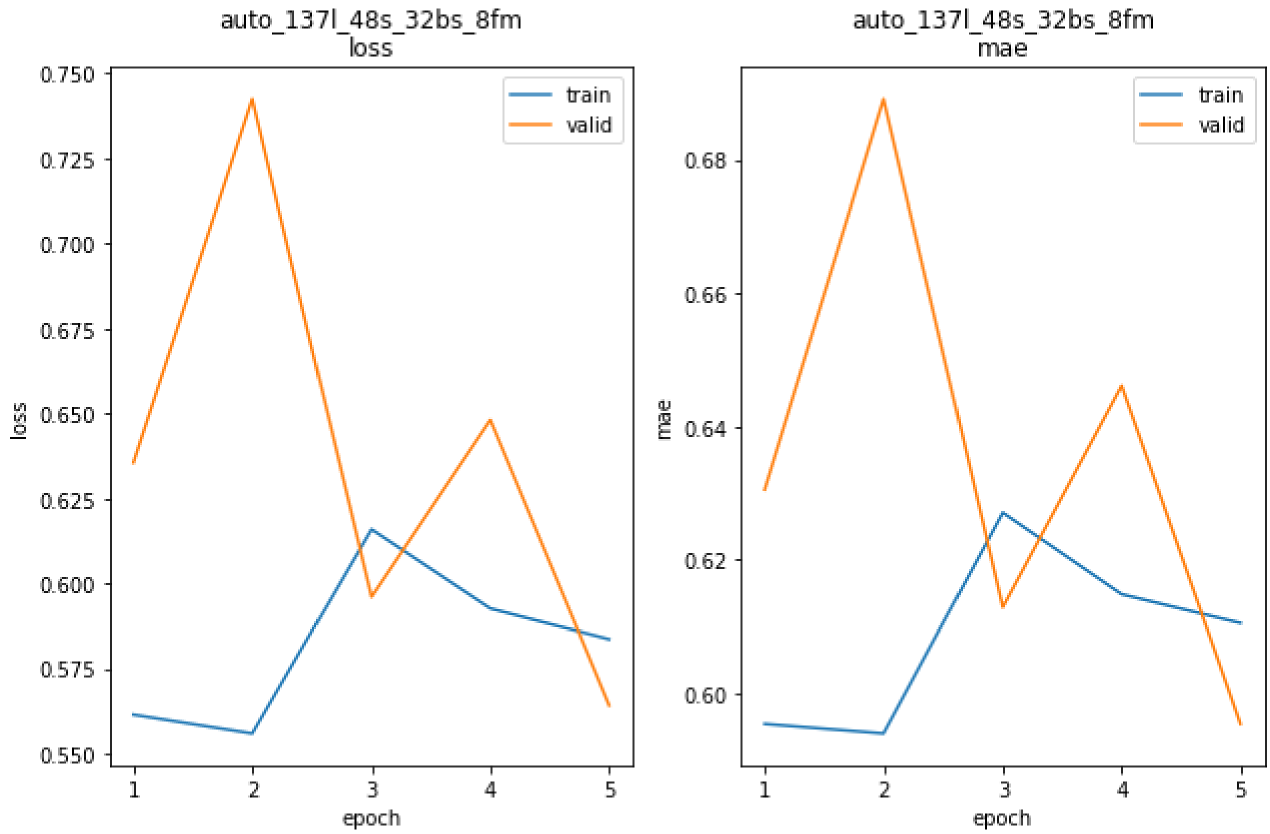



```
auto_31l_48s_32bs_48fm train min loss: 0.104485 mae: 0.243372   epoch: 5
auto_31l_48s_32bs_48fm valid min loss: 0.124914 mae: 0.260293   epoch: 5

auto_31l_48s_32bs_48fm
Iteration No: 7 ended. Evaluation done at random point.
Time taken: 236.8071
Function value obtained: 0.1249
Current minimum: 0.1226
Iteration No: 8 started. Evaluating function at random point.
lags 137
feat_maps 8
Epoch 1/5
5876/5876 [==============================] - 92s 15ms/step - loss: 0.3420 - ma
Epoch 2/5
5876/5876 [==============================] - 9s 1ms/step - loss: 1.2786 - mae:
```



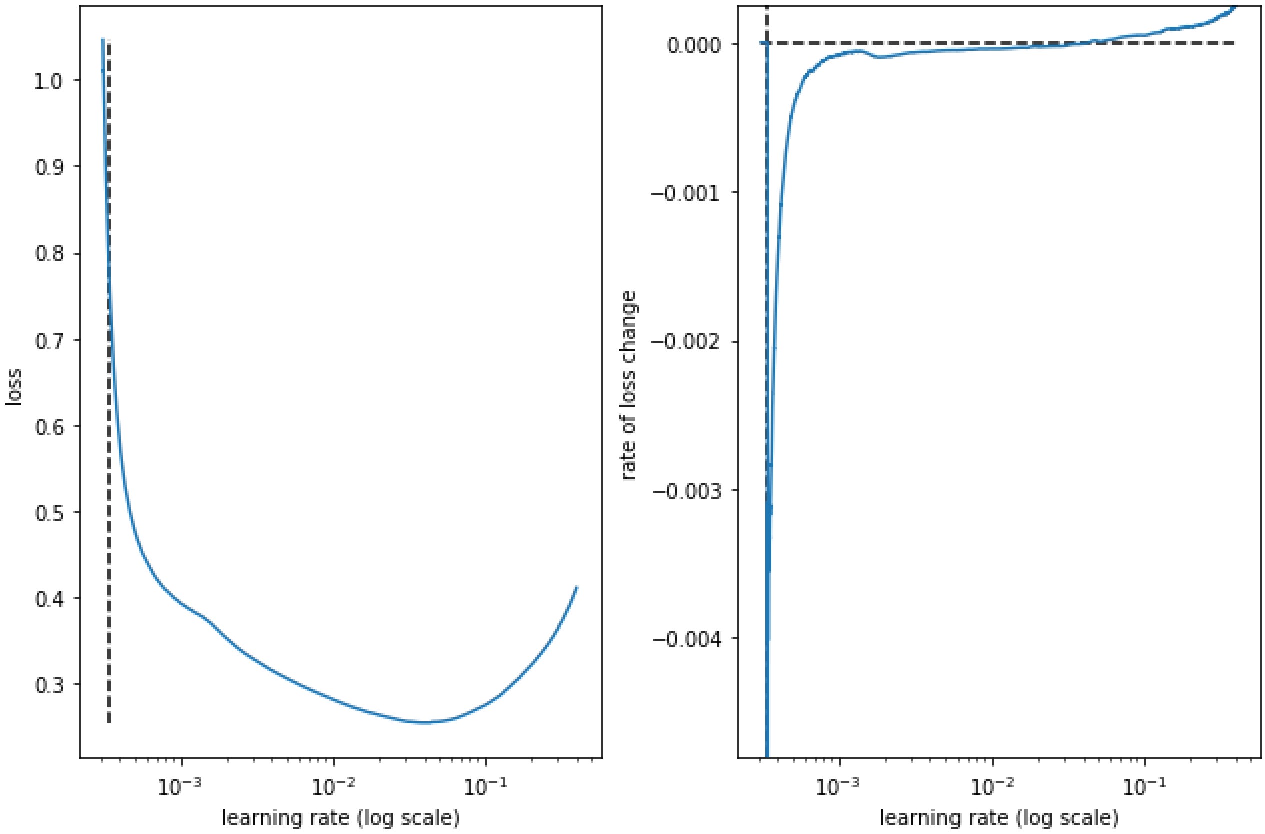



```
best lr: 0.7614389

Model: "auto_137l_48s_32bs_8fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 137, 10)]         0

 encoder (Functional)        (None, 137, 8)            608

 decoder (Functional)        (None, 48)                984

=================================================================
Total params: 1,592
Trainable params: 1,592
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5876/5876 - 83s - loss: 0.5615 - mae: 0.5954 - val_loss: 0.6355 - val_mae: 0.6
Epoch 2/5
5876/5876 - 80s - loss: 0.5560 - mae: 0.5940 - val_loss: 0.7424 - val_mae: 0.6
Epoch 3/5
5876/5876 - 80s - loss: 0.6159 - mae: 0.6271 - val_loss: 0.5961 - val_mae: 0.6
Epoch 4/5
5876/5876 - 80s - loss: 0.5927 - mae: 0.6148 - val_loss: 0.6481 - val_mae: 0.6
Epoch 5/5
5876/5876 - 80s - loss: 0.5837 - mae: 0.6105 - val_loss: 0.5642 - val_mae: 0.5
```



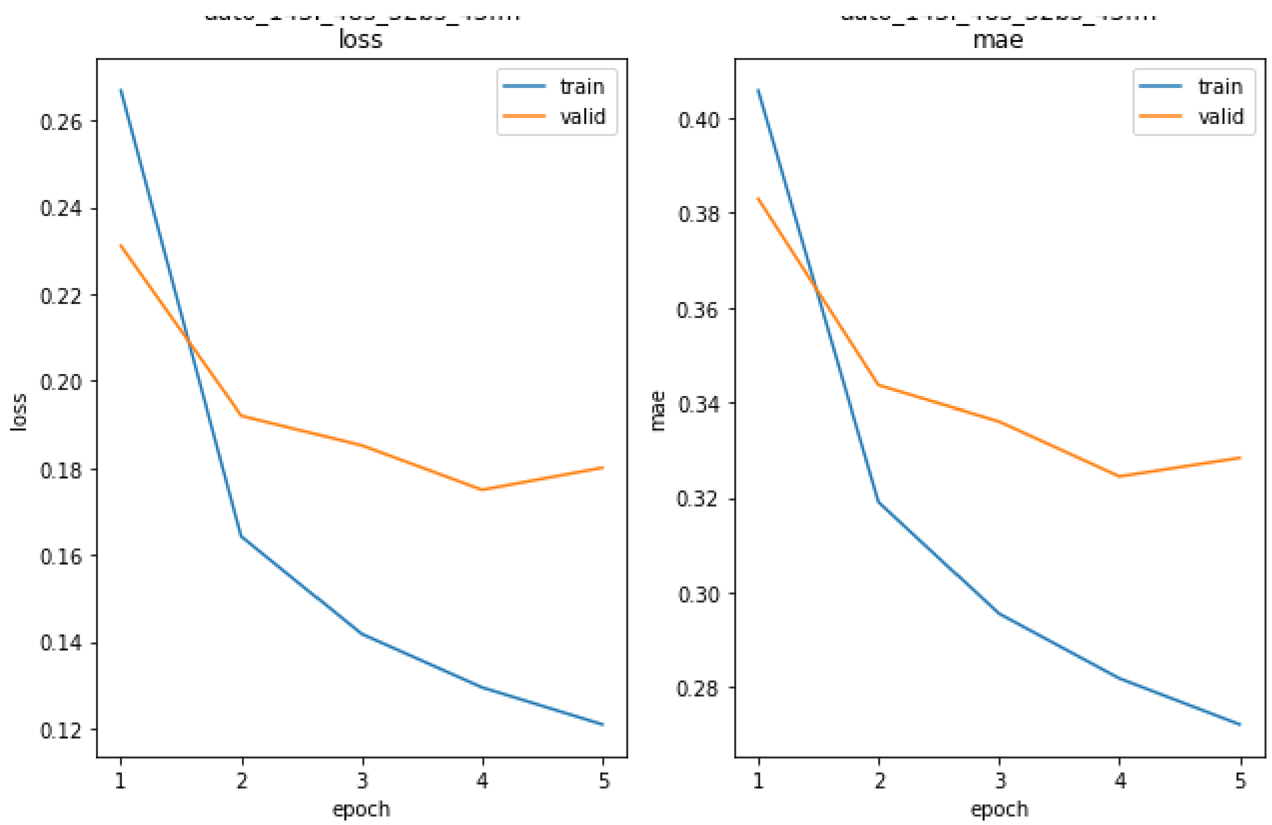

```
auto_137l_48s_32bs_8fm train min loss: 0.555998 mae: 0.594011   epoch: 2
auto_137l_48s_32bs_8fm valid min loss: 0.564245 mae: 0.595451   epoch: 5

auto_137l_48s_32bs_8fm
WARN: bad model auto_137l_48s_32bs_8fm
Iteration No: 8 ended. Evaluation done at random point.
```

```
Time taken: 504.5783
Function value obtained: 0.5642
Current minimum: 0.1226
Iteration No: 9 started. Evaluating function at random point.
lags 143
feat_maps 43
Epoch 1/5
5876/5876 [==============================] - 132s 22ms/step - loss: nan - mae:
```

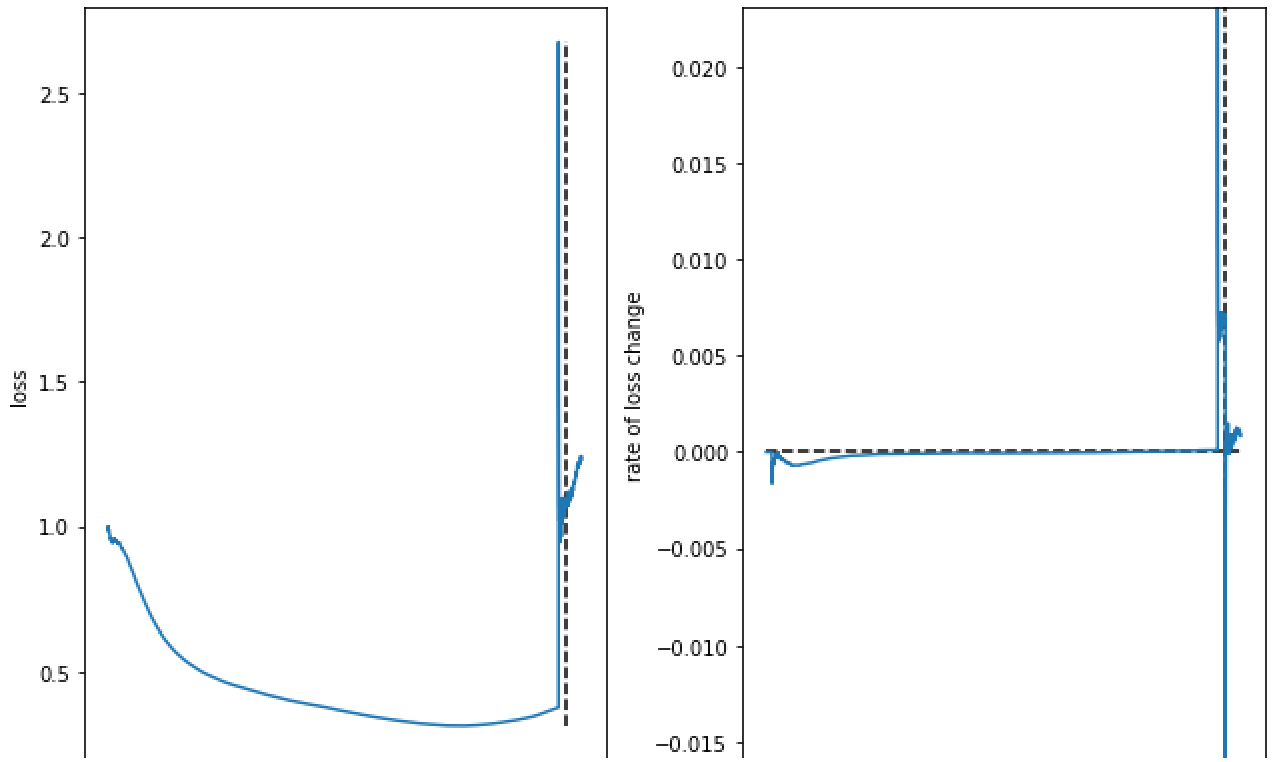



```
best lr: 0.00034293454

Model: "auto_143l_48s_32bs_43fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 143, 10)]         0

 encoder (Functional)        (None, 143, 43)           9288

 decoder (Functional)        (None, 48)                17119

=================================================================
Total params: 26,407
Trainable params: 26,407
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5876/5876 - 132s - loss: 0.2669 - mae: 0.4058 - val_loss: 0.2311 - val_mae: 0.
Epoch 2/5
5876/5876 - 129s - loss: 0.1643 - mae: 0.3190 - val_loss: 0.1921 - val_mae: 0.
Epoch 3/5
5876/5876 - 129s - loss: 0.1418 - mae: 0.2955 - val_loss: 0.1853 - val_mae: 0.
Epoch 4/5
5876/5876 - 129s - loss: 0.1295 - mae: 0.2818 - val_loss: 0.1750 - val_mae: 0.
Epoch 5/5
5876/5876 - 129s - loss: 0.1210 - mae: 0.2721 - val_loss: 0.1801 - val_mae: 0.
```

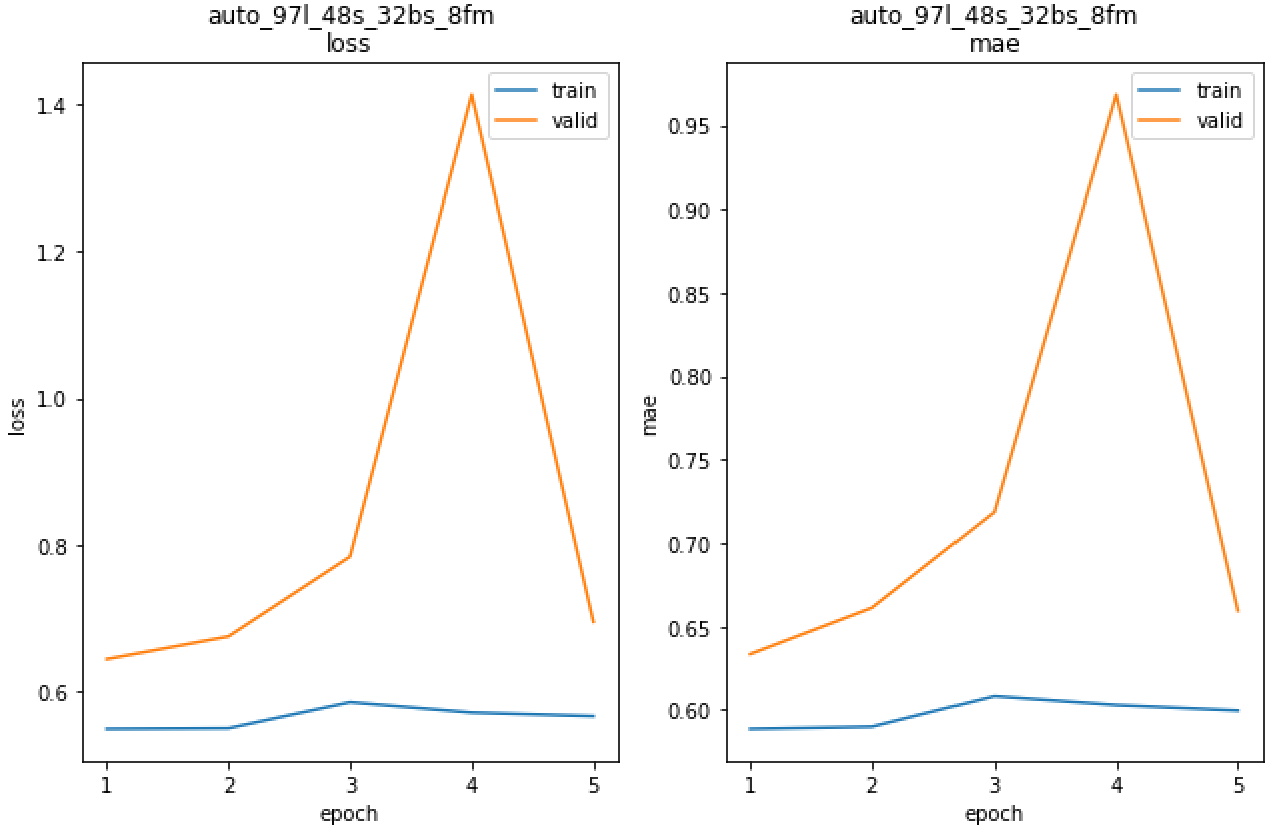
auto_143l_48s_32bs_43fm auto_143l_48s_32bs_43fm




```
auto_1431_48s_32bs_43fm train min loss: 0.120971        mae: 0.272135    epoch:
auto_1431_48s_32bs_43fm valid min loss: 0.175020        mae: 0.324456    epoch:

auto_1431_48s_32bs_43fm
Iteration No: 9 ended. Evaluation done at random point.
Time taken: 781.3246
Function value obtained: 0.1750
Current minimum: 0.1226
Iteration No: 10 started. Evaluating function at random point.
lags 97
feat_maps 8
Epoch 1/5
5878/5878 [==============================] - 72s 12ms/step - loss: 0.3773 - ma
Epoch 2/5
5878/5878 [==============================] - 4s 606us/step - loss: 1.2557 - ma
```

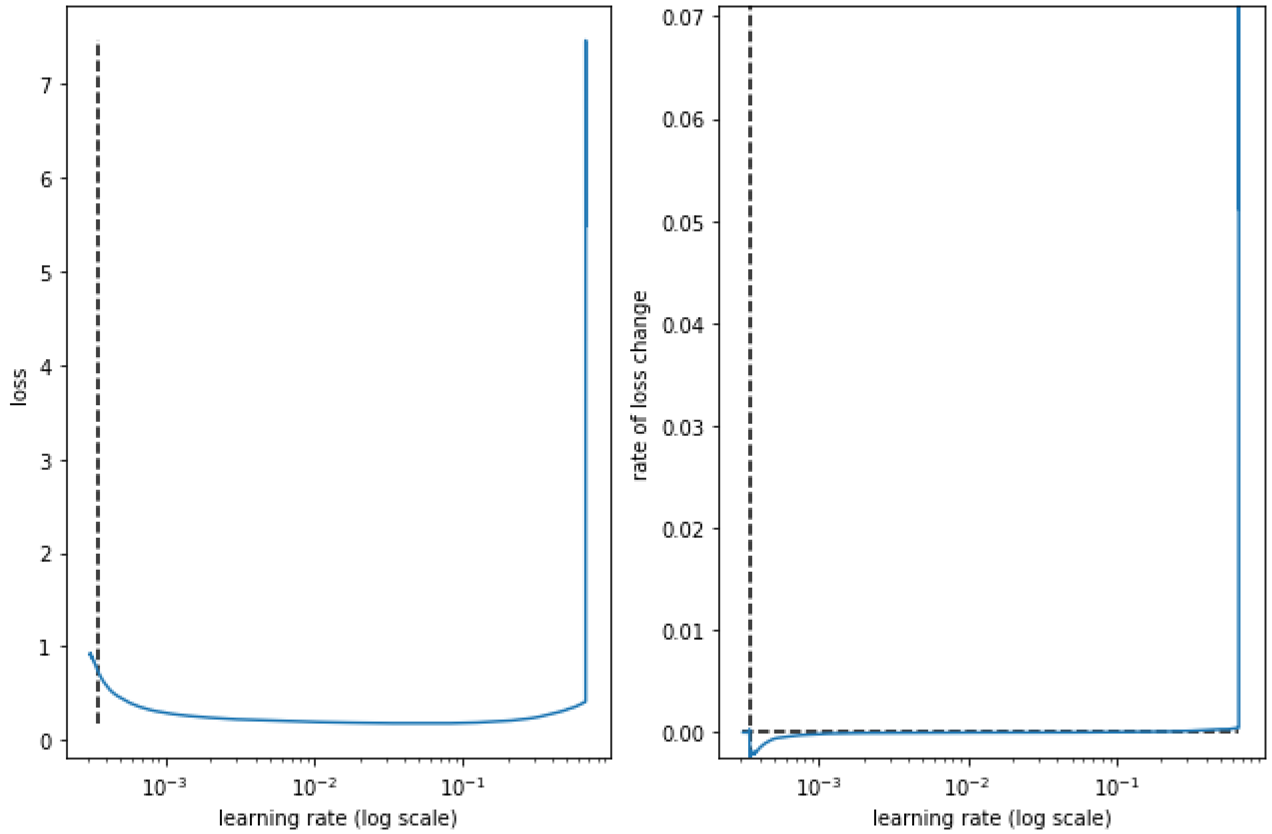



```
best lr: 0.75941336

Model: "auto_97l_48s_32bs_8fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 97, 10)]          0

 encoder (Functional)        (None, 97, 8)             608

 decoder (Functional)        (None, 48)                984

=================================================================
Total params: 1,592
Trainable params: 1,592
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5878/5878 - 62s - loss: 0.5483 - mae: 0.5886 - val_loss: 0.6437 - val_mae: 0.6
Epoch 2/5
5878/5878 - 59s - loss: 0.5492 - mae: 0.5900 - val_loss: 0.6747 - val_mae: 0.6
Epoch 3/5
5878/5878 - 59s - loss: 0.5848 - mae: 0.6083 - val_loss: 0.7838 - val_mae: 0.7
Epoch 4/5
5878/5878 - 59s - loss: 0.5707 - mae: 0.6030 - val_loss: 1.4137 - val_mae: 0.9
Epoch 5/5
5878/5878 - 59s - loss: 0.5658 - mae: 0.5997 - val_loss: 0.6947 - val_mae: 0.6
```

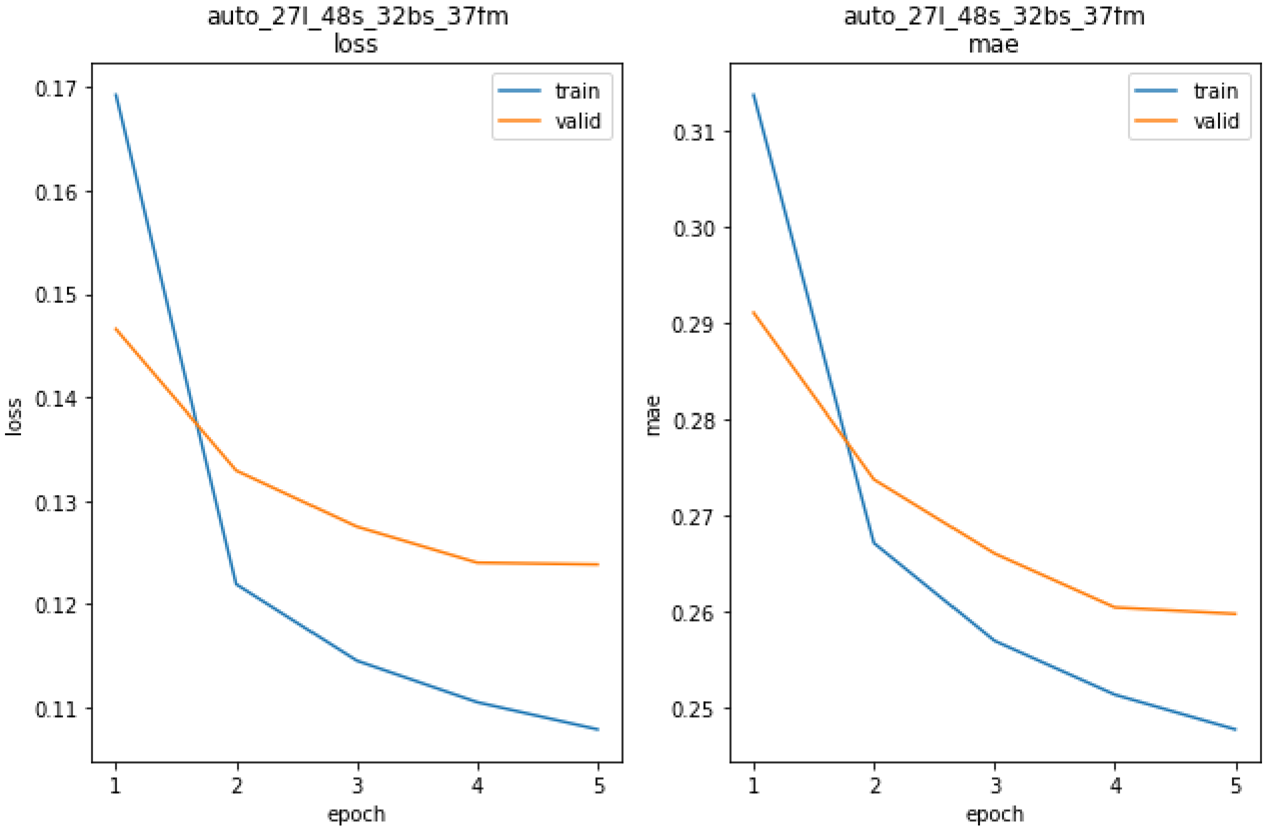



```
auto_97l_48s_32bs_8fm train min loss: 0.548308  mae: 0.588634   epoch: 1
auto_97l_48s_32bs_8fm valid min loss: 0.643712  mae: 0.633528   epoch: 1

auto_97l_48s_32bs_8fm
Iteration No: 10 ended. Evaluation done at random point.
Time taken: 374.6709
Function value obtained: 0.6437
```

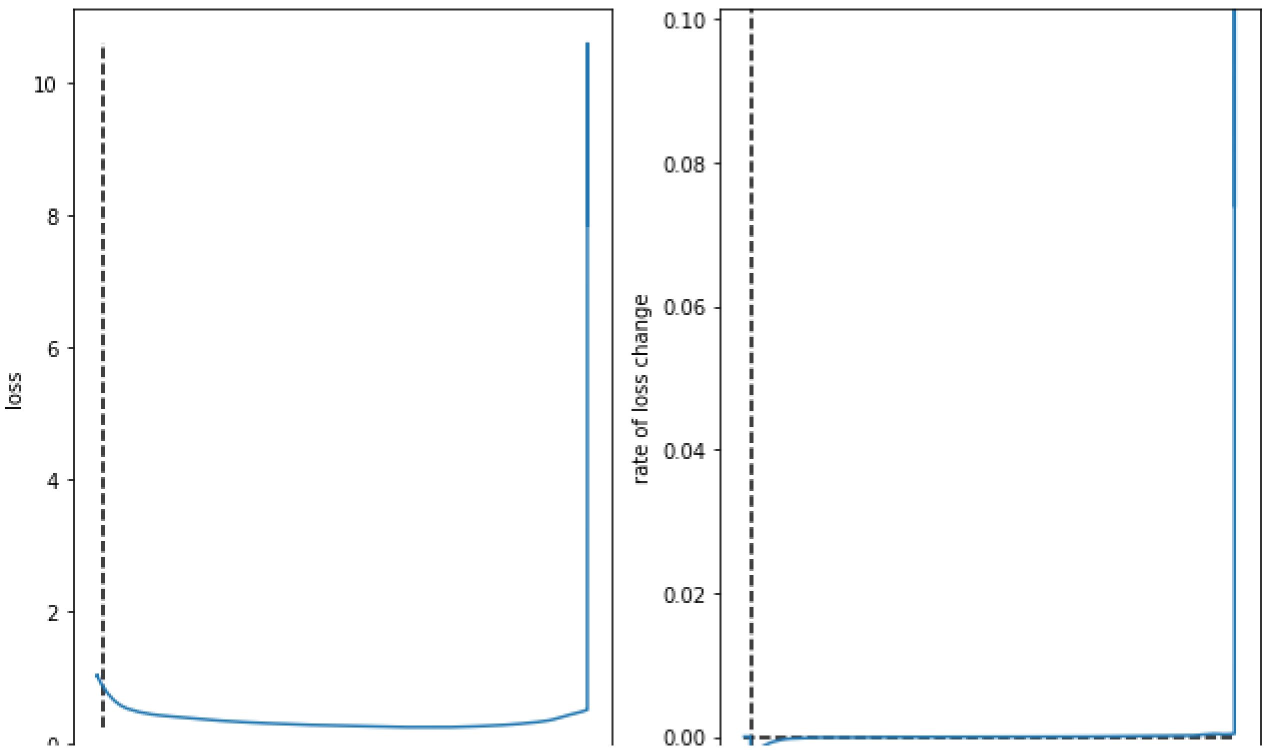
```
Function value obtained: 0.6437
Current minimum: 0.1226
Iteration No: 11 started. Evaluating function at random point.
lags 27
feat_maps 37
Epoch 1/5
5880/5880 [==============================] - 52s 8ms/step - loss: 0.4120 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 8us/step - loss: 2.6748 - mae:
```

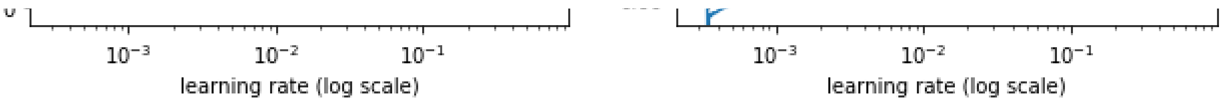



```
best lr: 0.0003469713

Model: "auto_27l_48s_32bs_37fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 27, 10)]          0

 encoder (Functional)        (None, 27, 37)            7104

 decoder (Functional)        (None, 48)                12961

=================================================================
Total params: 20,065
Trainable params: 20,065
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 40s - loss: 0.1693 - mae: 0.3137 - val_loss: 0.1466 - val_mae: 0.2
Epoch 2/5
5880/5880 - 36s - loss: 0.1219 - mae: 0.2671 - val_loss: 0.1329 - val_mae: 0.2
Epoch 3/5
5880/5880 - 36s - loss: 0.1145 - mae: 0.2570 - val_loss: 0.1275 - val_mae: 0.2
Epoch 4/5
5880/5880 - 36s - loss: 0.1105 - mae: 0.2514 - val_loss: 0.1240 - val_mae: 0.2
Epoch 5/5
5880/5880 - 36s - loss: 0.1079 - mae: 0.2477 - val_loss: 0.1239 - val_mae: 0.2
```

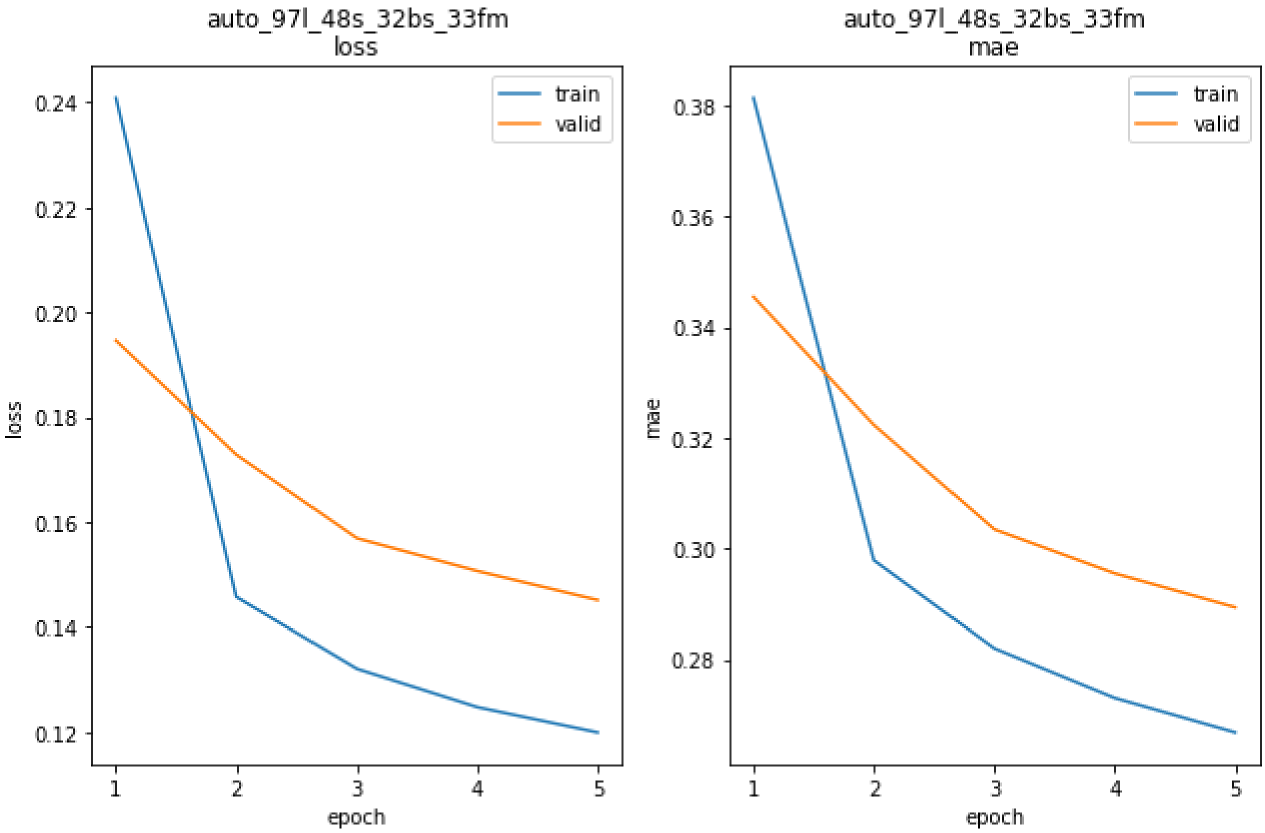



```
auto_27l_48s_32bs_37fm train min loss: 0.107885 mae: 0.247748   epoch: 5
auto_27l_48s_32bs_37fm valid min loss: 0.123851 mae: 0.259763   epoch: 5

auto_27l_48s_32bs_37fm
Iteration No: 11 ended. Evaluation done at random point.
Time taken: 244.0861
Function value obtained: 0.1239
Current minimum: 0.1226
Iteration No: 12 started. Evaluating function at random point.
lags 97
feat_maps 33
Epoch 1/5
5878/5878 [==============================] - 90s 15ms/step - loss: 0.5052 - ma
Epoch 2/5
5878/5878 [==============================] - 0s 15us/step - loss: 4.8514 - mae
```



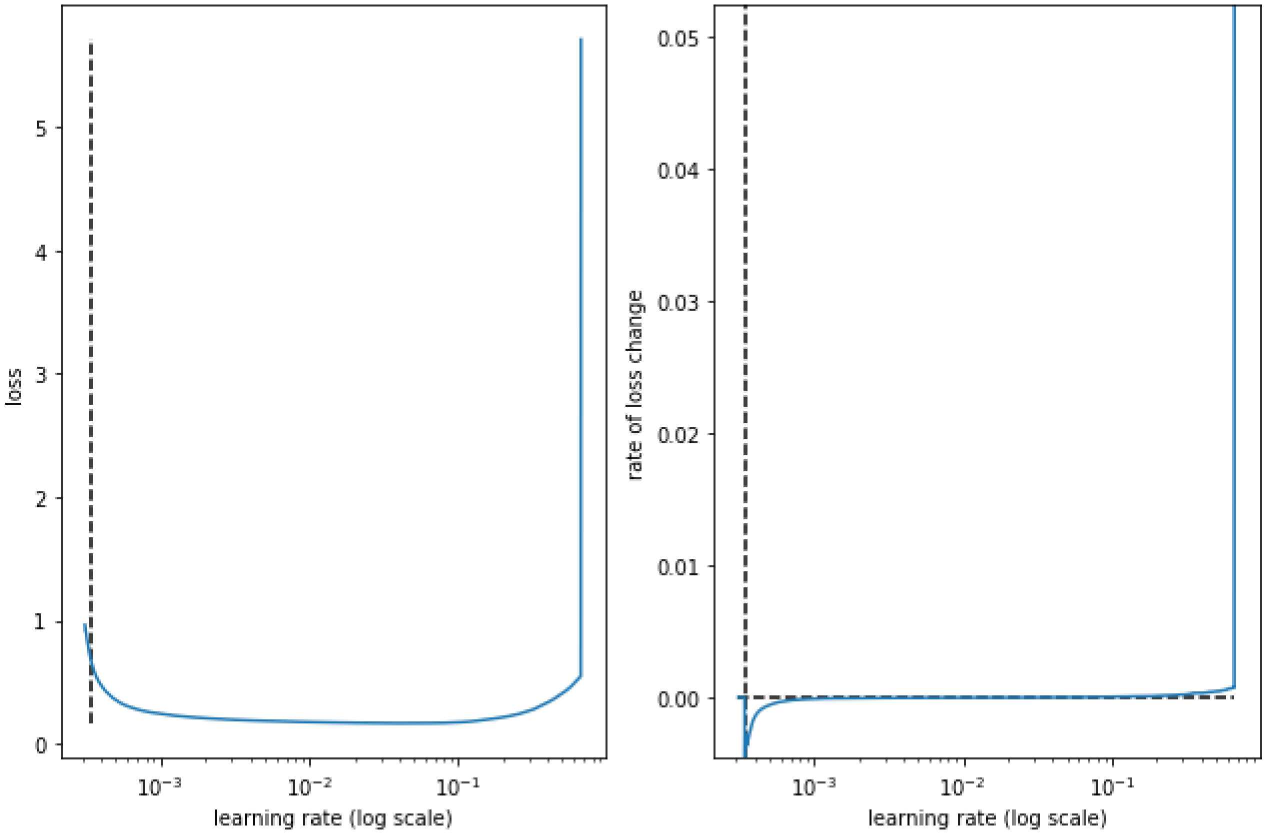



```
best lr: 0.00034291876

Model: "auto_97l_48s_32bs_33fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 97, 10)]          0

 encoder (Functional)        (None, 97, 33)            5808

 decoder (Functional)        (None, 48)                10509

=================================================================
Total params: 16,317
Trainable params: 16,317
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5878/5878 - 81s - loss: 0.2408 - mae: 0.3814 - val_loss: 0.1946 - val_mae: 0.3
Epoch 2/5
5878/5878 - 78s - loss: 0.1458 - mae: 0.2979 - val_loss: 0.1729 - val_mae: 0.3
Epoch 3/5
5878/5878 - 78s - loss: 0.1321 - mae: 0.2820 - val_loss: 0.1569 - val_mae: 0.3
Epoch 4/5
5878/5878 - 78s - loss: 0.1248 - mae: 0.2730 - val_loss: 0.1507 - val_mae: 0.2
Epoch 5/5
5878/5878 - 78s - loss: 0.1200 - mae: 0.2668 - val_loss: 0.1451 - val_mae: 0.2
```

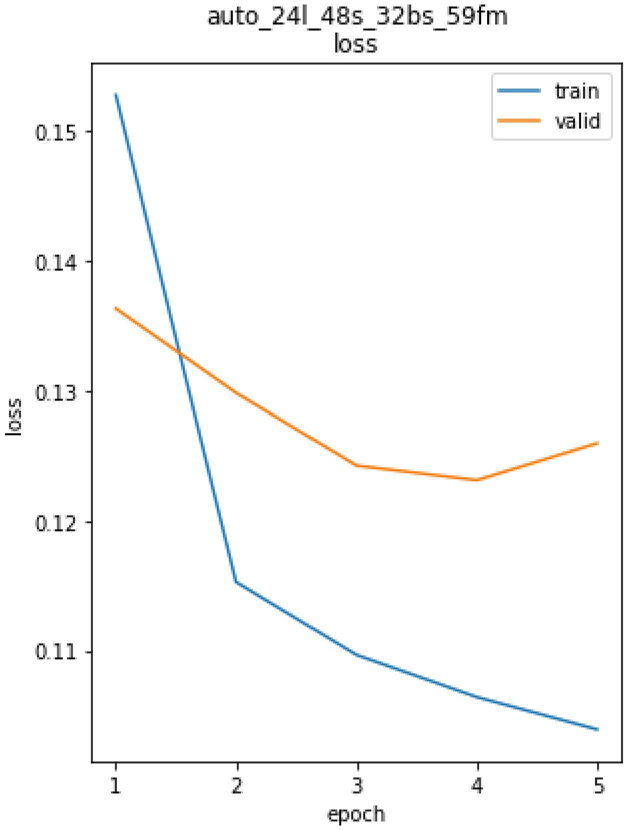


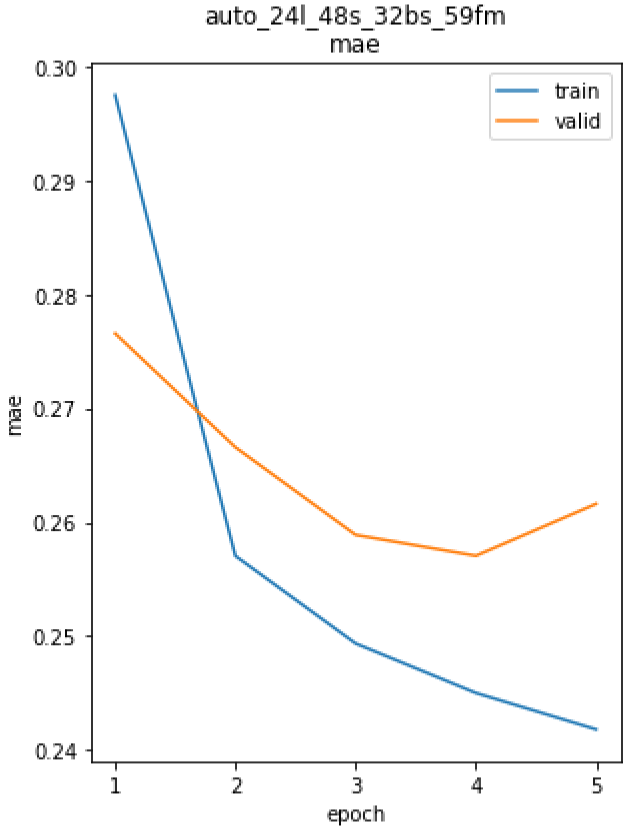

```
auto_97l_48s_32bs_33fm train min loss: 0.119963 mae: 0.266822    epoch: 5
auto_97l_48s_32bs_33fm valid min loss: 0.145141 mae: 0.289388    epoch: 5

auto_97l_48s_32bs_33fm
Iteration No: 12 ended. Evaluation done at random point.
Time taken: 484.8562
```

```
Function value obtained: 0.1451
Current minimum: 0.1226
Iteration No: 13 started. Searching for the next optimal point.
lags 24
feat_maps 59
Epoch 1/5
5880/5880 [==============================] - 50s 8ms/step - loss: 0.5482 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 8us/step - loss: 5.8920 - mae:
```

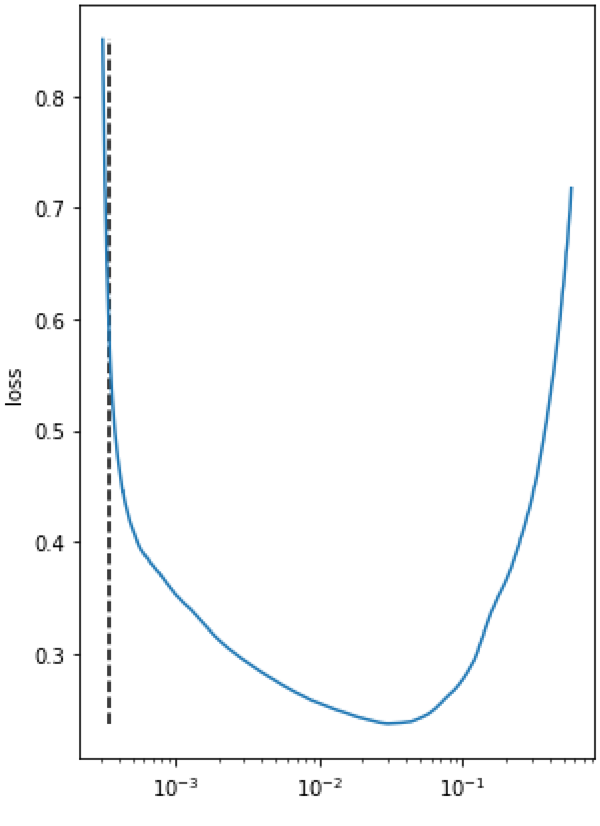


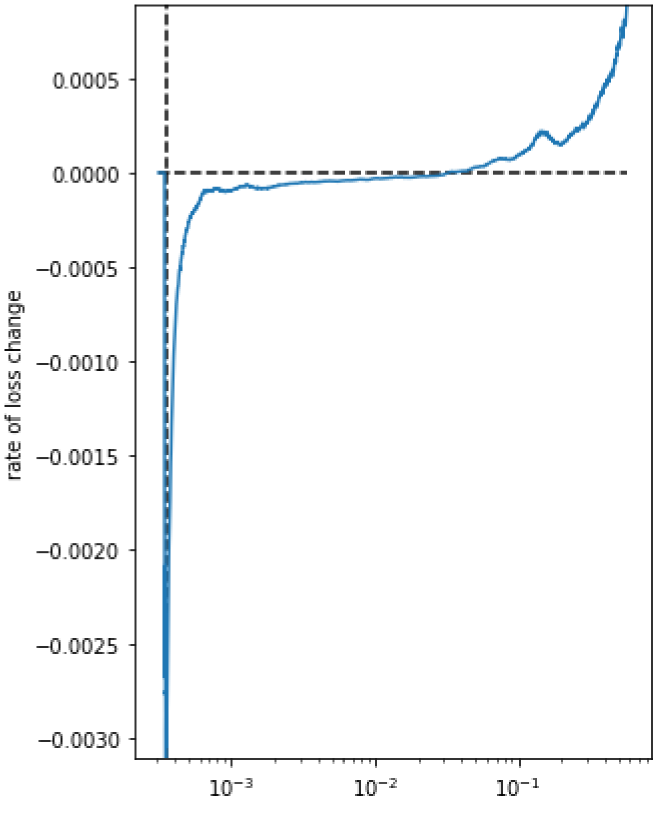

```
best lr: 0.00034200563

Model: "auto_24l_48s_32bs_59fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_13 (InputLayer)       [(None, 24, 10)]          0

 encoder (Functional)        (None, 24, 59)            16520

 decoder (Functional)        (None, 48)                31023

=================================================================
Total params: 47,543
Trainable params: 47,543
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 40s - loss: 0.1528 - mae: 0.2975 - val_loss: 0.1363 - val_mae: 0.2
Epoch 2/5
5880/5880 - 36s - loss: 0.1153 - mae: 0.2570 - val_loss: 0.1298 - val_mae: 0.2
Epoch 3/5
5880/5880 - 36s - loss: 0.1097 - mae: 0.2493 - val_loss: 0.1243 - val_mae: 0.2
Epoch 4/5
5880/5880 - 36s - loss: 0.1064 - mae: 0.2450 - val_loss: 0.1231 - val_mae: 0.2
Epoch 5/5
5880/5880 - 36s - loss: 0.1039 - mae: 0.2418 - val_loss: 0.1260 - val_mae: 0.2
```



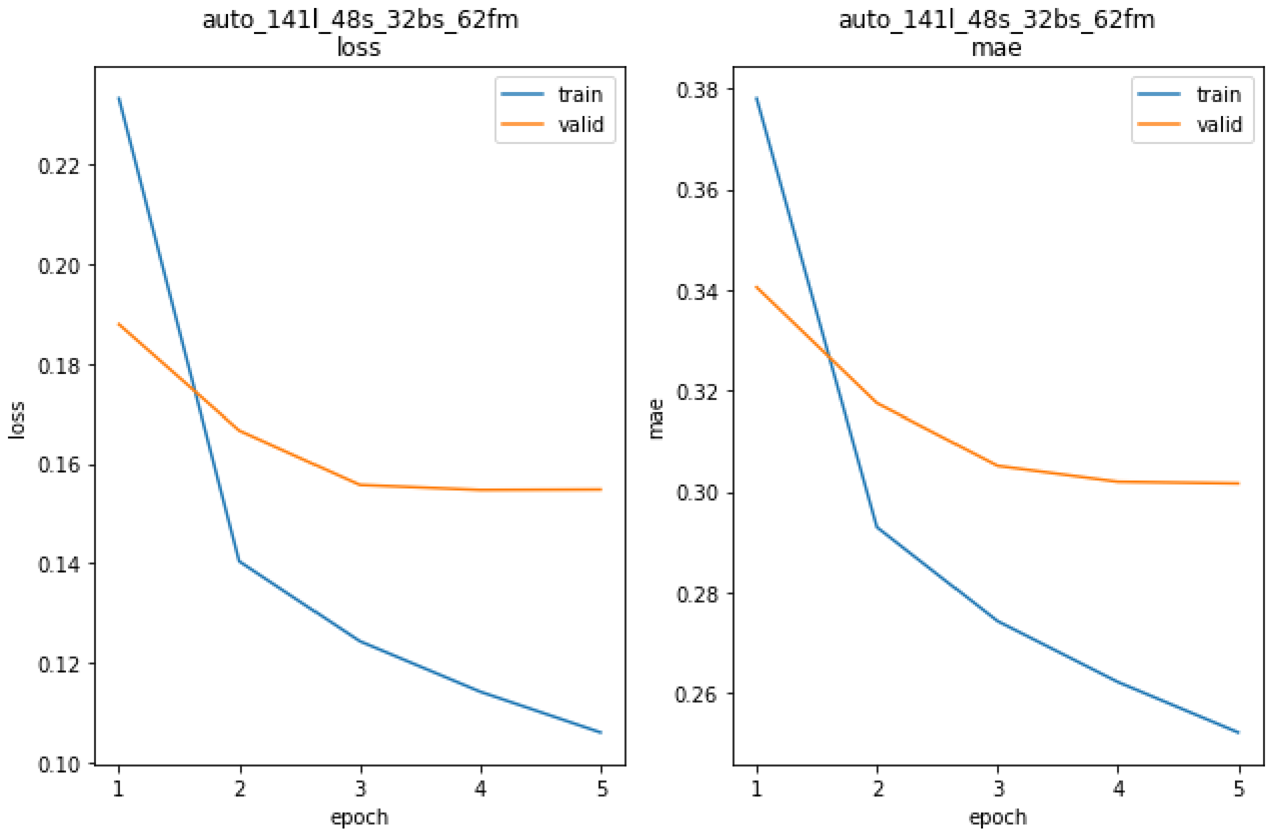



```
auto_24l_48s_32bs_59fm train min loss: 0.103946 mae: 0.241803   epoch: 5
auto_24l_48s_32bs_59fm valid min loss: 0.123137 mae: 0.257052   epoch: 4

auto_24l_48s_32bs_59fm
Iteration No: 13 ended. Search finished for the next optimal point.
Time taken: 242.1410
Function value obtained: 0.1231
Current minimum: 0.1226
Iteration No: 14 started. Searching for the next optimal point.
lags 141
feat_maps 62
Epoch 1/5
5876/5876 [==============================] - 161s 27ms/step - loss: nan - mae:
```



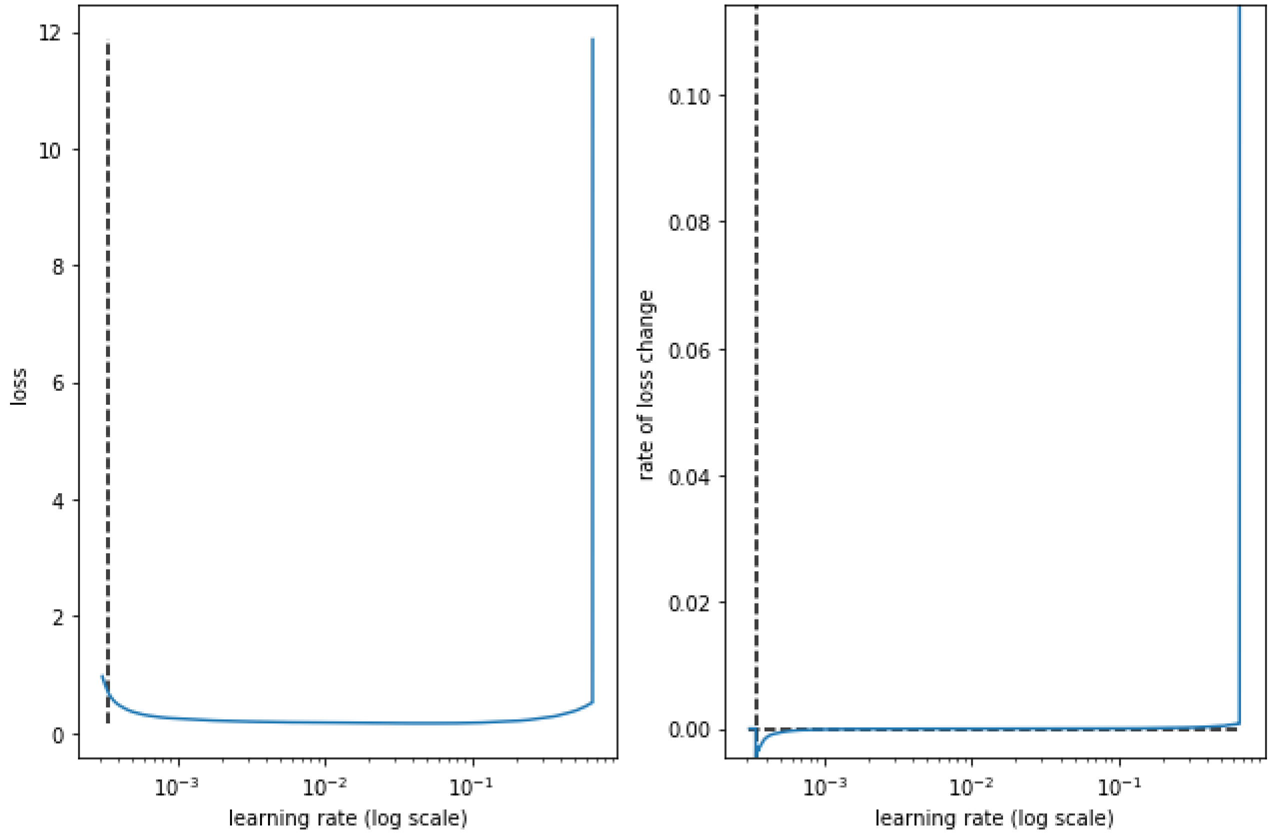



```
best lr: 0.00034837364

Model: "auto_141l_48s_32bs_62fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_14 (InputLayer)       [(None, 141, 10)]         0

 encoder (Functional)        (None, 141, 62)           18104

 decoder (Functional)        (None, 48)                34086

=================================================================
Total params: 52,190
Trainable params: 52,190
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5876/5876 - 153s - loss: 0.2333 - mae: 0.3780 - val_loss: 0.1879 - val_mae: 0.
Epoch 2/5
5876/5876 - 150s - loss: 0.1404 - mae: 0.2930 - val_loss: 0.1666 - val_mae: 0.
Epoch 3/5
5876/5876 - 150s - loss: 0.1244 - mae: 0.2743 - val_loss: 0.1558 - val_mae: 0.
Epoch 4/5
5876/5876 - 150s - loss: 0.1142 - mae: 0.2622 - val_loss: 0.1547 - val_mae: 0.
Epoch 5/5
5876/5876 - 150s - loss: 0.1060 - mae: 0.2522 - val_loss: 0.1548 - val_mae: 0.
```




```
auto_141l_48s_32bs_62fm train min loss: 0.106041        mae: 0.252189    epoch:
auto_141l_48s_32bs_62fm valid min loss: 0.154735        mae: 0.301989    epoch:

auto_141l_48s_32bs_62fm
Iteration No: 14 ended. Search finished for the next optimal point.
Time taken: 968.6429
Function value obtained: 0.1547
Current minimum: 0.1226
```

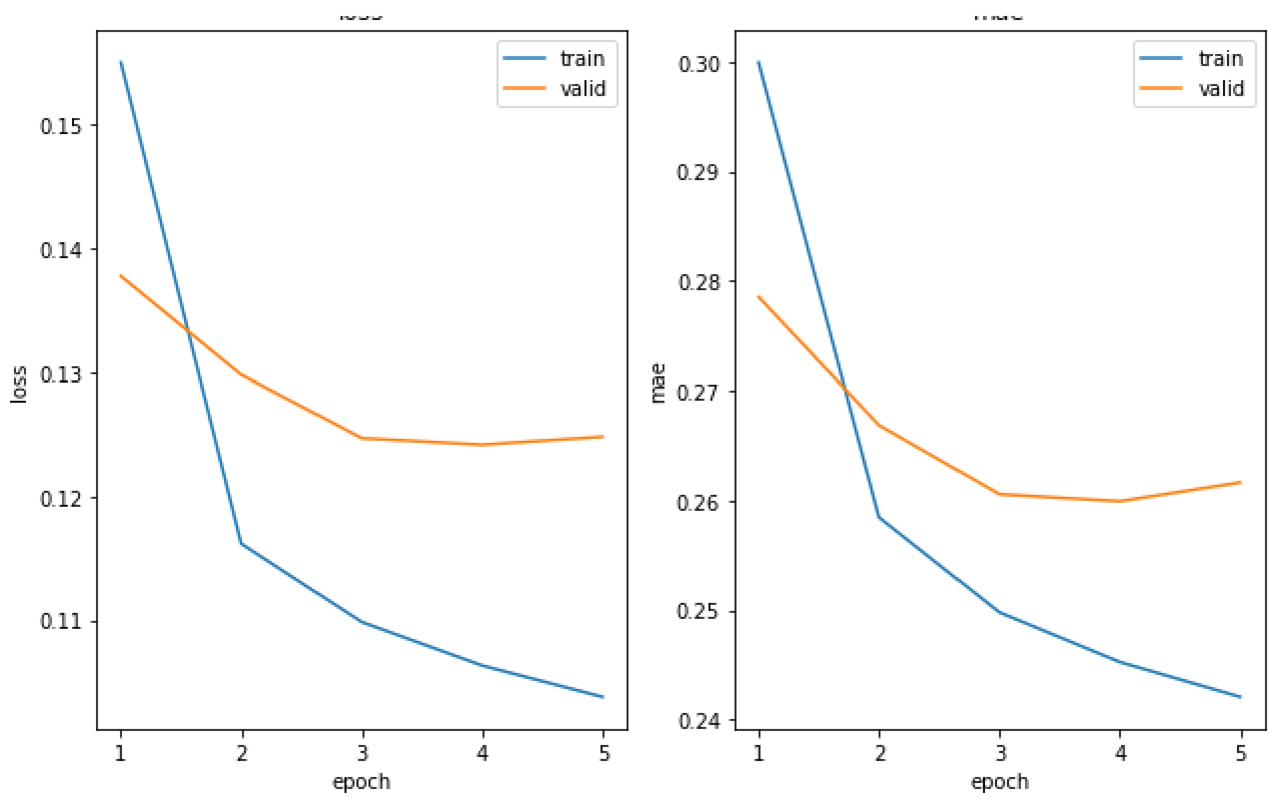
```
Iteration No: 15 started. Searching for the next optimal point.
lags 24
feat_maps 54
Epoch 1/5
5880/5880 [==============================] - 51s 8ms/step - loss: 0.5254 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 9us/step - loss: 6.5072 - mae:
```



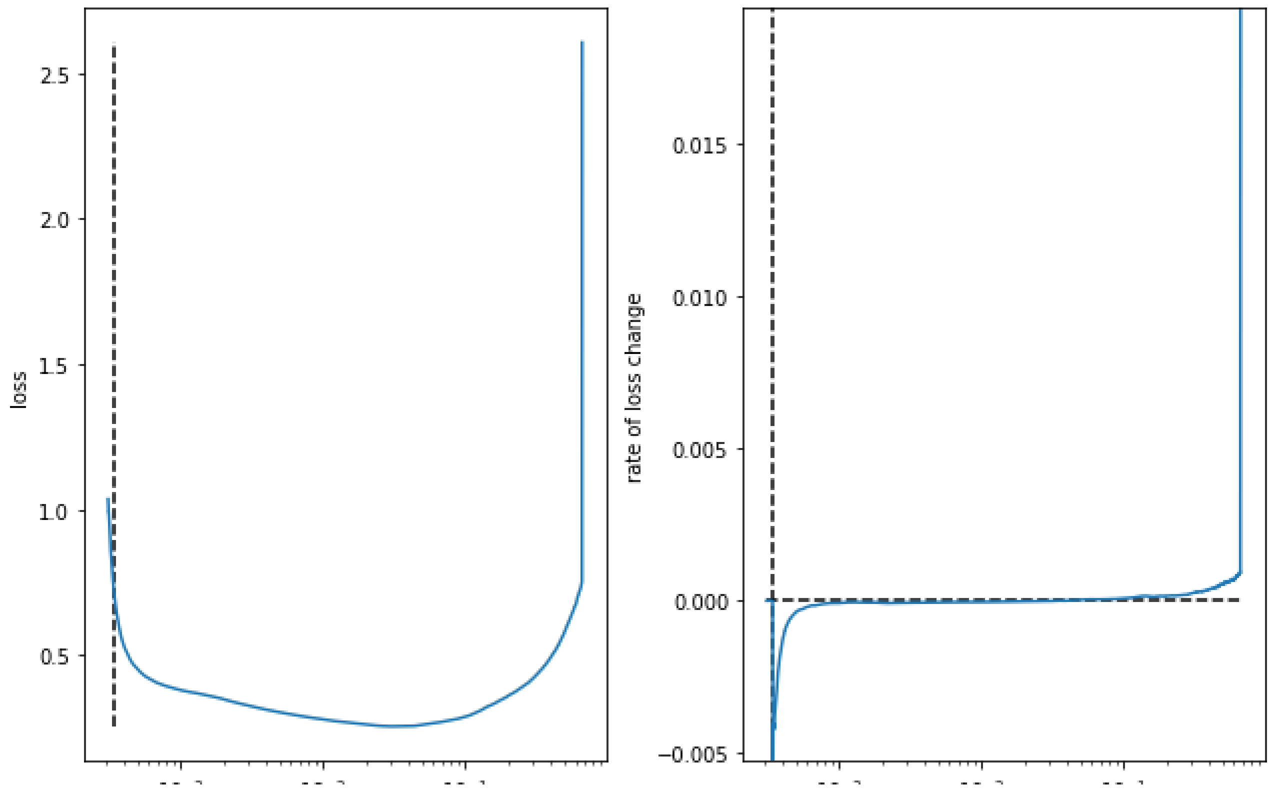

```
best lr: 0.00034200563

Model: "auto_24l_48s_32bs_54fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_15 (InputLayer)       [(None, 24, 10)]          0

 encoder (Functional)        (None, 24, 54)            14040

 decoder (Functional)        (None, 48)                26238

=================================================================
Total params: 40,278
Trainable params: 40,278
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 39s - loss: 0.1550 - mae: 0.2999 - val_loss: 0.1377 - val_mae: 0.2
Epoch 2/5
5880/5880 - 35s - loss: 0.1162 - mae: 0.2585 - val_loss: 0.1298 - val_mae: 0.2
Epoch 3/5
5880/5880 - 35s - loss: 0.1098 - mae: 0.2498 - val_loss: 0.1247 - val_mae: 0.2
Epoch 4/5
5880/5880 - 35s - loss: 0.1063 - mae: 0.2453 - val_loss: 0.1242 - val_mae: 0.2
Epoch 5/5
5880/5880 - 36s - loss: 0.1038 - mae: 0.2421 - val_loss: 0.1248 - val_mae: 0.2
```

auto_24l_48s_32bs_54fm loss

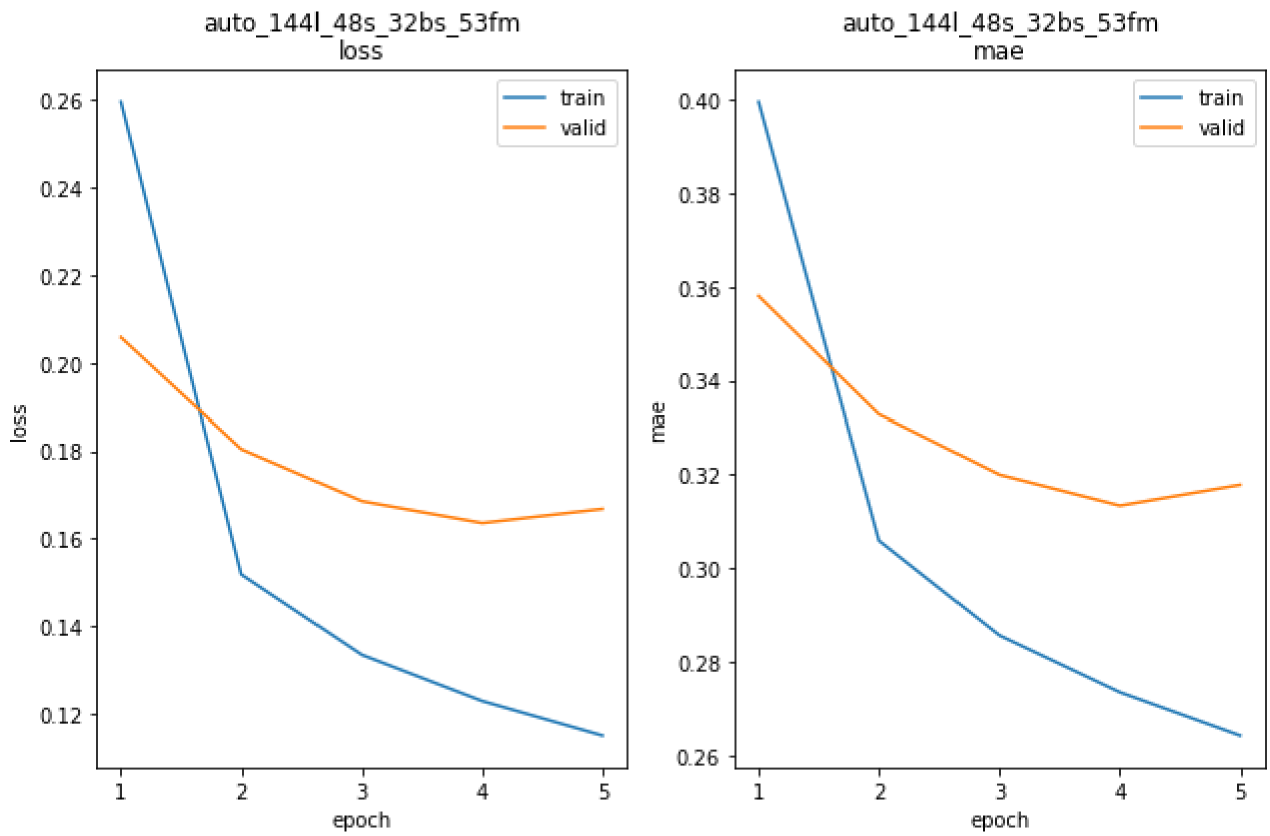
auto_24l_48s_32bs_54fm mae




```
auto_24l_48s_32bs_54fm train min loss: 0.103810 mae: 0.242090   epoch: 5
auto_24l_48s_32bs_54fm valid min loss: 0.124157 mae: 0.259946   epoch: 4

auto_24l_48s_32bs_54fm
Iteration No: 15 ended. Search finished for the next optimal point.
Time taken: 269.1141
Function value obtained: 0.1242
Current minimum: 0.1226
Iteration No: 16 started. Searching for the next optimal point.
lags 144
feat_maps 53
Epoch 1/5
5876/5876 [==============================] - 154s 26ms/step - loss: 0.7491 - m
Epoch 2/5
5876/5876 [==============================] - 0s 26us/step - loss: 2.8116 - mae
```

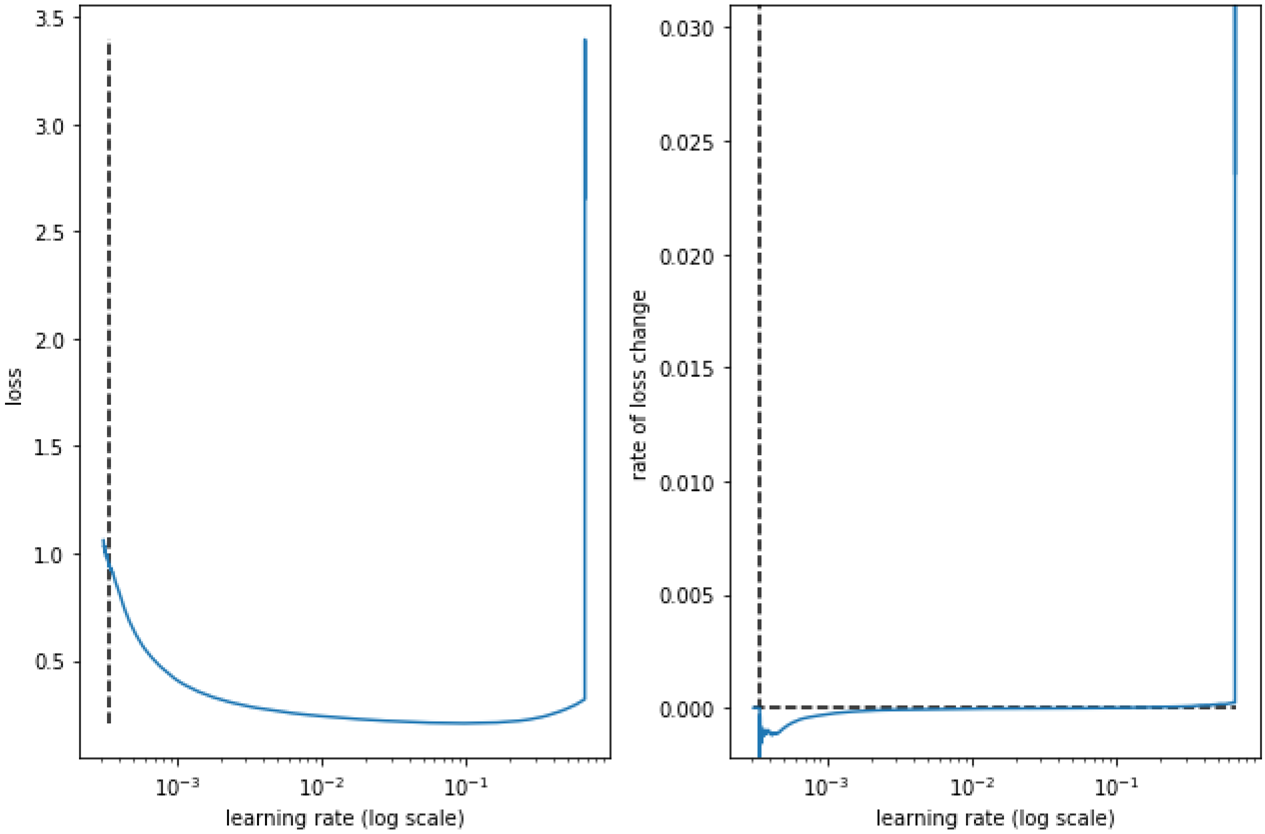



```
best lr: 0.00034293454

Model: "auto_144l_48s_32bs_53fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_16 (InputLayer)       [(None, 144, 10)]         0

 encoder (Functional)        (None, 144, 53)           13568

 decoder (Functional)        (None, 48)                25329

=================================================================
Total params: 38,897
Trainable params: 38,897
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5876/5876 - 145s - loss: 0.2595 - mae: 0.3996 - val_loss: 0.2058 - val_mae: 0.
Epoch 2/5
5876/5876 - 140s - loss: 0.1519 - mae: 0.3059 - val_loss: 0.1804 - val_mae: 0.
Epoch 3/5
5876/5876 - 140s - loss: 0.1335 - mae: 0.2856 - val_loss: 0.1686 - val_mae: 0.
Epoch 4/5
5876/5876 - 140s - loss: 0.1230 - mae: 0.2734 - val_loss: 0.1636 - val_mae: 0.
Epoch 5/5
5876/5876 - 140s - loss: 0.1151 - mae: 0.2642 - val_loss: 0.1668 - val_mae: 0.
```



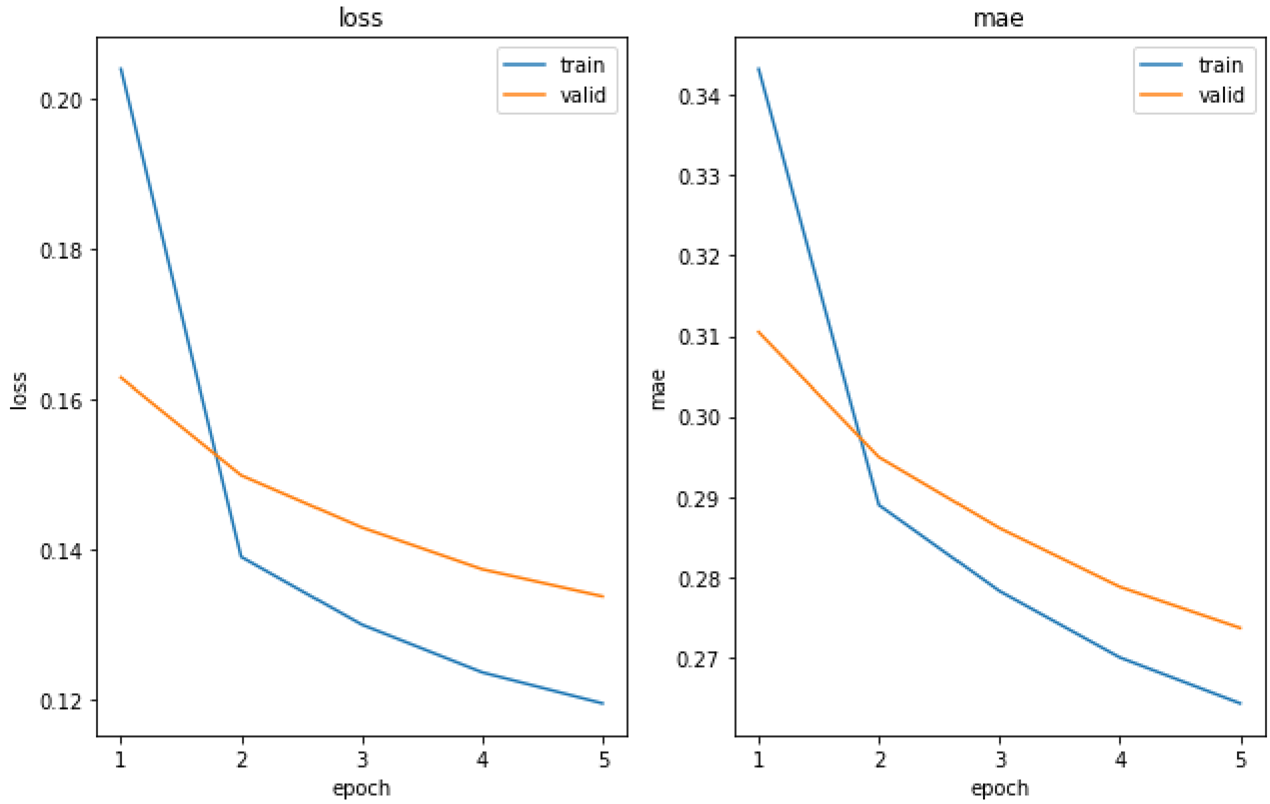

```
auto_144l_48s_32bs_53fm train min loss: 0.115077        mae: 0.264188    epoch:
auto_144l_48s_32bs_53fm valid min loss: 0.163587        mae: 0.313373    epoch:

auto_144l_48s_32bs_53fm
Iteration No: 16 ended. Search finished for the next optimal point.
Time taken: 863.1181
Function value obtained: 0.1636
```

```
Current minimum: 0.1226
Iteration No: 17 started. Searching for the next optimal point.
lags 24
feat_maps 17
Epoch 1/5
5880/5880 [==============================] - 51s 8ms/step - loss: 0.3191 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 8us/step - loss: 1.5924 - mae:
```




```
best lr: 0.00034200563

Model: "auto_24l_48s_32bs_17fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_17 (InputLayer)       [(None, 24, 10)]          0

 encoder (Functional)        (None, 24, 17)            1904

 decoder (Functional)        (None, 48)                3261

=================================================================
Total params: 5,165
Trainable params: 5,165
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
5880/5880 - 39s - loss: 0.2040 - mae: 0.3432 - val_loss: 0.1628 - val_mae: 0.3
Epoch 2/5
5880/5880 - 36s - loss: 0.1390 - mae: 0.2890 - val_loss: 0.1499 - val_mae: 0.2
Epoch 3/5
5880/5880 - 36s - loss: 0.1299 - mae: 0.2783 - val_loss: 0.1429 - val_mae: 0.2
Epoch 4/5
5880/5880 - 35s - loss: 0.1236 - mae: 0.2700 - val_loss: 0.1373 - val_mae: 0.2
Epoch 5/5
5880/5880 - 36s - loss: 0.1194 - mae: 0.2643 - val_loss: 0.1337 - val_mae: 0.2
```

auto_24l_48s_32bs_17fm auto_24l_48s_32bs_17fm




```
auto_24l_48s_32bs_17fm train min loss: 0.119449 mae: 0.264268   epoch: 5
auto_24l_48s_32bs_17fm valid min loss: 0.133718 mae: 0.273663   epoch: 5

auto_24l_48s_32bs_17fm
Iteration No: 17 ended. Search finished for the next optimal point.
Time taken: 235.4074
Function value obtained: 0.1337
Current minimum: 0.1226
Iteration No: 18 started. Searching for the next optimal point.
lags 24
feat_maps 40
Epoch 1/5
5880/5880 [==============================] - 51s 8ms/step - loss: 0.4447 - mae
Epoch 2/5
5880/5880 [==============================] - 0s 9us/step - loss: 3.3713 - mae:
```