LSTM Networks for Cambridge UK Weather Time Series

LSTM 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 include results up to 24 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

for further details including:

- data description
- data cleaning and preparation
- data exploration
- baseline models

In particular, see the keras_mlp_fcn_resnet_time_series notebook, which uses a streamlined version of data preparation from Tensorflow time series forecasting tutorial. That notebook showed promising results for LSTM networks.

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

```
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 prev

#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 InputLayer, Dense, Dropout, Activation, \
                         Flatten, Reshape, LSTM, RepeatVector, Conv1D, \
                         TimeDistributed, Bidirectional, Dropout, \
                         MaxPooling1D, MaxPooling2D, Conv2D, ConvLSTM1D  # TODO Re
from keras.layers.merge import concatenate
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
Tue Jun 21 15:08:20 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   37C    P0    33W / 250W |    375MiB / 16280MiB |      3%      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/CamMetCleanish2021.04.26.csv"
df = pd.read_csv(data_loc, parse_dates = True)

df['ds'] = pd.to_datetime(df['ds'])
df.set_index(df['ds'], drop = False, inplace = True)
df = df[~df.index.duplicated(keep = 'first')]

df['y'] = df['y'] / 10
df['wind.speed.mean'] = df['wind.speed.mean'] / 10

df = df.loc[df['ds'] > '2008-08-01 00:00:00',]
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.speed.mean', 'wind.bearing.mean']
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, 7)

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 7 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
dtypes: datetime64[ns](1), float64(6)
memory usage: 13.6 MB
None

Summary stats:
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCc
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCc
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  if sys.path[0] == '':
```

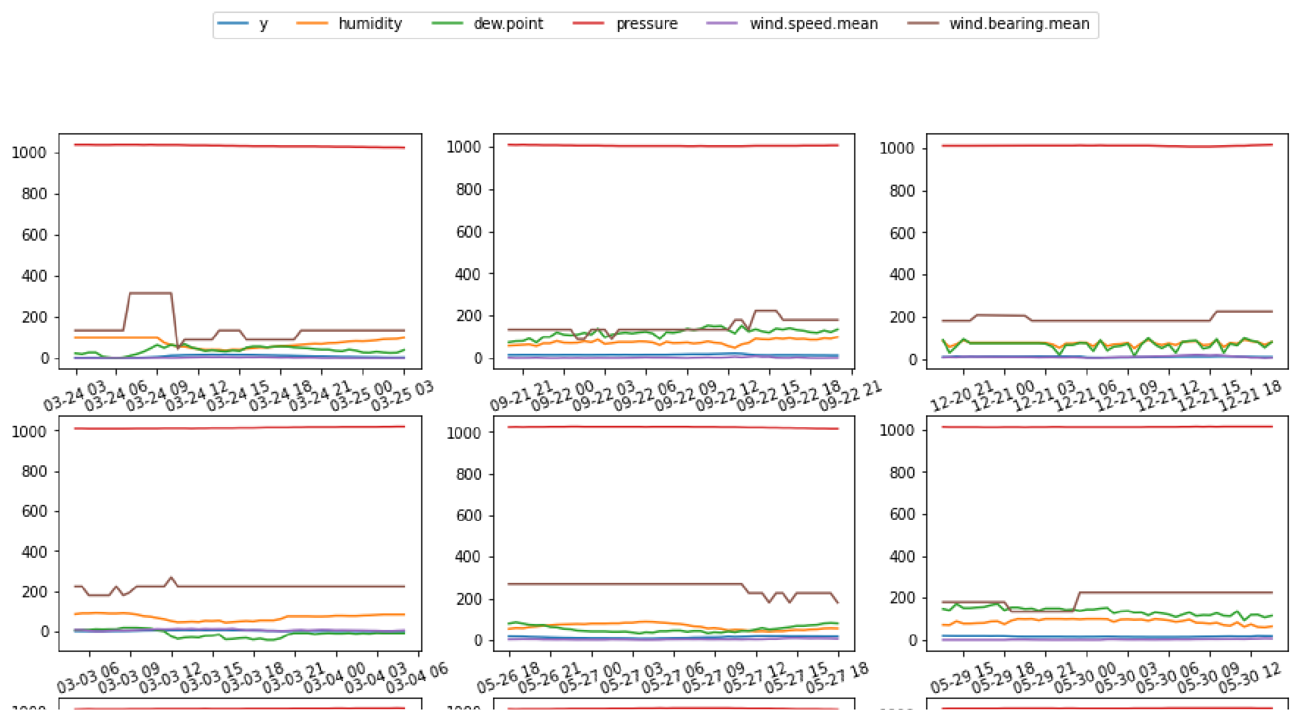
	y	humidity	dew.point	pressure	wind.speed.mean	w
count	223250.000000	223250.000000	223250.000000	223250.000000	223250.000000	
mean	10.000512	78.689959	58.880634	1014.336135	4.432390	
std	6.496255	17.274417	51.630120	11.935364	4.013553	
min	-7.000000	20.000000	-100.000000	963.000000	0.000000	
25%	5.200000	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-	2008-						

01 01:00:00	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 × 7 columns



Data Processing and Feature Engineering

The data must be reformatted before model building.

The following steps are carried out:

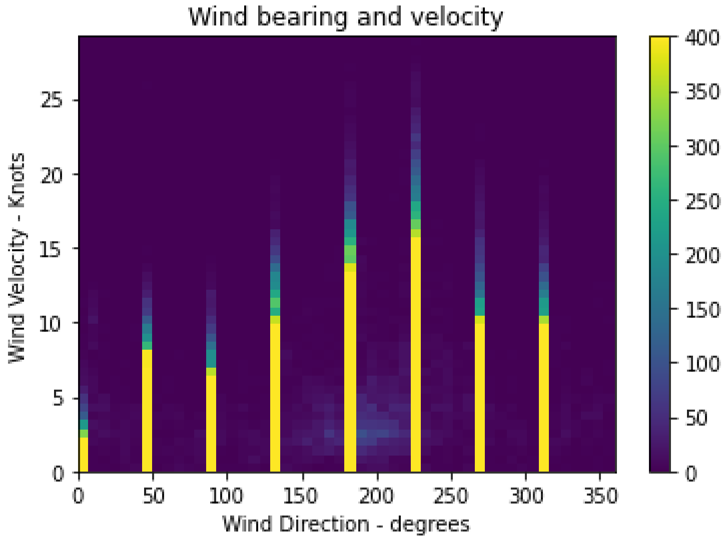
- Wind direction and speed transformation
- Time conversion
- TBATS seasonal components addition
- Train test data separation
- Data normalisation
- Data windowing

Wind direction and speed transformation

The `wind.bearing.mean` column gives wind direction in degrees but is categorised at 45 degree increments, i.e. 0, 45, 90, 135, 180, 225, 270, 315. Wind direction shouldn't matter if the wind is not blowing.

The distribution of wind direction and speed looks like this:

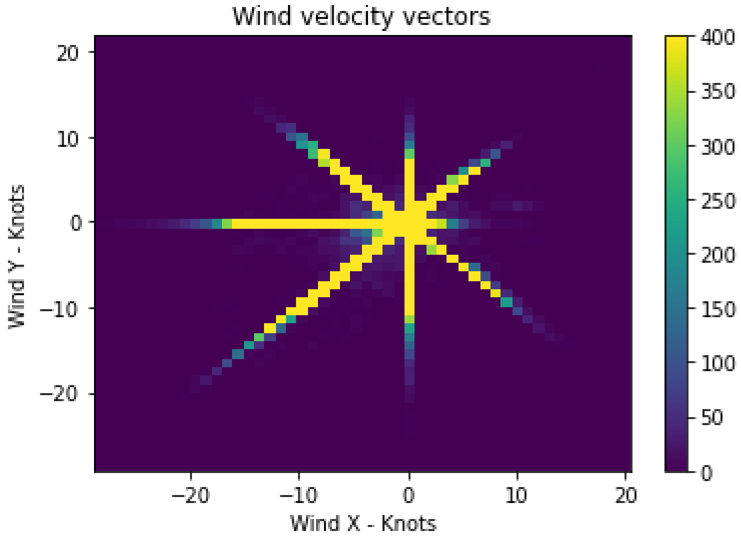
```
plt.hist2d(df['wind.bearing.mean'], df['wind.speed.mean'], bins = (50, 50), vmax =
plt.colorbar()
plt.xlabel('Wind Direction - degrees')
plt.ylabel('Wind Velocity - Knots')
plt.title('Wind bearing and velocity');
```



Convert wind direction and speed to x and y vectors, so the model can more easily interpret them.

```
wv = df['wind.speed.mean']

# Convert to radians
```

```
wd_rad = df['wind.bearing.mean'] * np.pi / 180

# Calculate the wind x and y components
df['wind.x'] = wv * np.cos(wd_rad)
df['wind.y'] = wv * np.sin(wd_rad)

df_orig = df

plt.hist2d(df['wind.x'], df['wind.y'], bins = (50, 50), vmax = 400)
plt.colorbar()
plt.xlabel('Wind X - Knots')
plt.ylabel('Wind Y - Knots')
plt.title('Wind velocity vectors');
```



Wind velocity vectors are better, but are still 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.

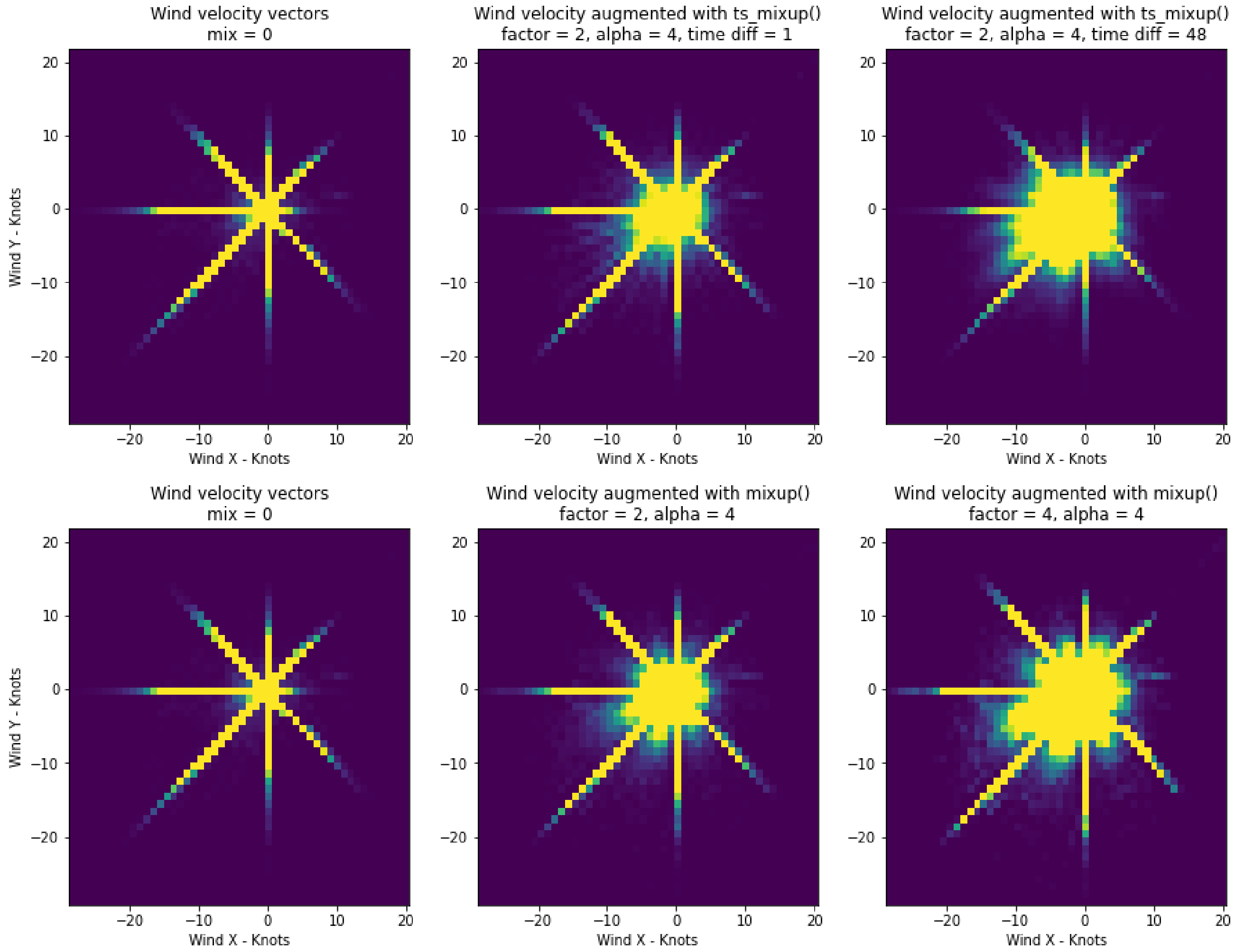
Here is a comparison of the improvement in wind velocity sparsity with standard mixup augmentation and a time series specific mixup.

```
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)
```



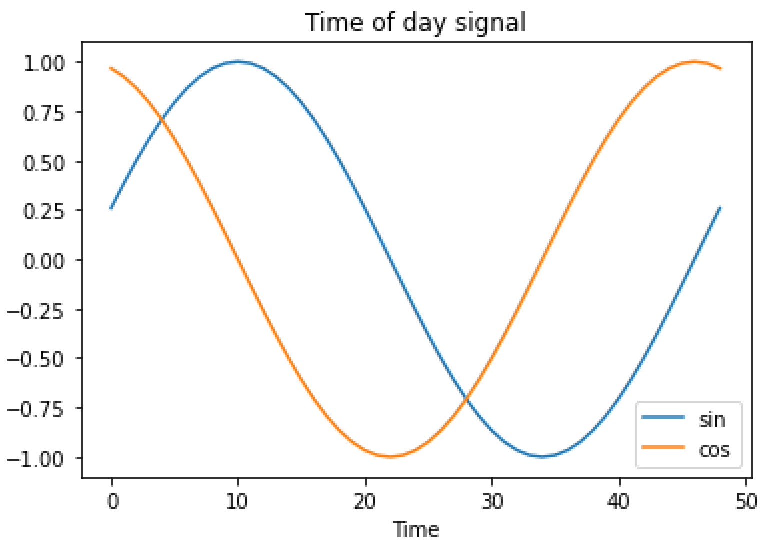
Mixup improves the categorical legacy of the wind velocity data. Unfortunately, if outliers are present their influence may be reinforced. A priori it's difficult to say which mixup variant is preferable.

Time conversion

Convert `ds` timestamps to "time of day" and "time of year" variables using `sin` and `cos` functions.

```
# Convert to secs
date_time   = pd.to_datetime(df['ds'], format = '%Y.%m.%d %H:%M:%S')
timestamp_s = date_time.map(datetime.datetime.timestamp)
```

```
day  = 24 * 60 * 60
year = 365.2425 * day

df['day.sin']  = np.sin(timestamp_s * (2 * np.pi / day))
df['day.cos']  = np.cos(timestamp_s * (2 * np.pi / day))
df['year.sin'] = np.sin(timestamp_s * (2 * np.pi / year))
df['year.cos'] = np.cos(timestamp_s * (2 * np.pi / year))

plt.plot(np.array(df['day.sin'])[49:98])
plt.plot(np.array(df['day.cos'])[49:98])
plt.xlabel('Time')
plt.legend(['sin', 'cos'], loc = 'lower right')
plt.title('Time of day signal');


# For use in other notebooks
if not 'google.colab' in str(get_ipython()):
    data_loc = "../data/CamMetPrepped2021.04.26.csv"
    df.to_csv(data_loc)
```



The yearly time components may benefit from a single phase shift so they align with the seasonal temperature peak around the end of July and temperature trough around the end of January. Similarly, the daily components may benefit from small daily phase shifts.

I implemented two approaches to acheive this:

1. TBATS seasonal components
2. Time2Vec representation
    - as this notebook is getting quite long I've removed the Time2Vec work
    - still available in [this commit]()

I also checked the following time component representations:

- [savgol_filter from scipy]()
- [lowess from statsmodels]()

- phase-shifted time components

These 3 methods described annual seasonality well but struggled with daily seasonality. Check the notebook commit history if interested.

The Short Time Fourier Transform (STFT) may be a good option for modeling the daily seasonality. It is available in scipy.

TBATS seasonal components - data preparation

The TBATS (exponential smoothing state space model with Box-Cox Transformation, ARMA errors, Trend and Seasonal components) method allows the seasonality to slowly change over time. It is a univariate method.

Time components were generated using the tbats functions from the forecast package. Some of the forecast package authors originated the TBATS method.

Python tbats implementations:

- sktime tbats function
- tbats package
- neither have functions for extracting seasonal components :-(

TBATS seasonal component generation code is here.

```
data_loc = "https://github.com/makeyourownmaker/CambridgeTemperatureModel/blob/mas

tbats = pd.read_csv(data_loc, parse_dates = True)
tbats = tbats.drop(columns='observed', axis=1)

tbats['level']   = tbats['level'] / 10
tbats['season1'] = tbats['season1'] / 10
tbats['season2'] = tbats['season2'] / 10

display(tbats)

df['doy']  = df.index.dayofyear
df['secs'] = ((df['ds'] - df['ds'].dt.normalize()) / pd.Timedelta('1 second')).ast

df = pd.merge(df, tbats,  how = 'left', left_on = ['doy', 'secs'], right_on = ['do
df = df.drop(columns='doy',  axis=1)
df = df.drop(columns='secs', axis=1)


df.set_index(df['ds'], drop = False, inplace = True)
df_orig = df

display(df.info())
display(df.describe())

display(df)
```

```
i = 1
cols = ['level', 'season2', 'season1']
plt.figure(figsize = (12, 6))
for col in cols:
    plt.subplot(len(cols), 1, i)
    plt.plot(df.loc[:, col])
    plt.title(col, y = 0.5, loc = 'right')
    i += 1
plt.show()

plt.figure(figsize = (12, 6))
plt.plot(df.loc[(df['ds'] > '2010-1-1') & (df['ds'] <= '2010-12-31'), 'season1'])
plt.title('season1 - single year', y = 0.5, loc = 'right')
plt.show()

plt.figure(figsize = (12, 6))
plt.plot(df.loc[(df['ds'] > '2010-1-1') & (df['ds'] <= '2010-1-22'), 'season1'])
plt.title('season1 - first 20 days', y = 0.5, loc = 'right')
plt.show()
```

	doy	secs	level	season1	season2
0	1	0	12.778523	-1.605236	-6.470564
1	1	1800	12.813697	-1.790321	-6.472173
2	1	3600	12.847716	-1.961594	-6.473835
3	1	5400	12.882007	-2.118536	-6.475488
4	1	7200	12.915066	-2.260283	-6.477141
...	...	...	...	...	...
17563	366	77400	8.812951	0.001560	-6.462863
17564	366	79200	8.837509	-0.157629	-6.464492
17565	366	81000	8.864565	-0.312037	-6.466079
17566	366	82800	8.877571	-0.460626	-6.467683
17567	366	84600	8.893226	-0.605462	-6.469183

17568 rows × 5 columns

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 223250 entries, 2008-08-01 00:30:00 to 2021-04-26 01:00:00
Data columns (total 16 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
 13  level              223250 non-null  float64
 14  season1            223250 non-null  float64
 15  season2            223250 non-null  float64
dtypes: datetime64[ns](1), float64(15)
memory usage: 29.0 MB
None
```

	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	

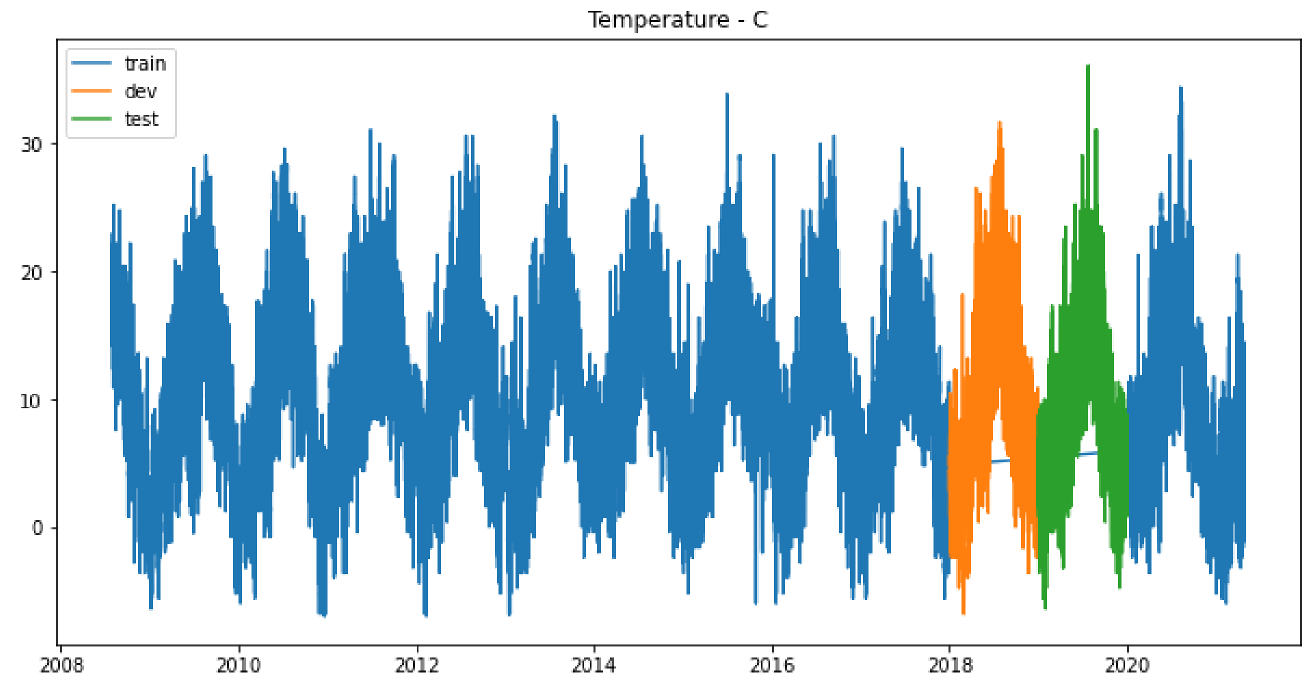
`season2` represents annual variation and `season1` represents daily variation over a 1 year period. `season1` is slowly changing throughout the year but is still sinusoidal. Unfortunately, the `season1` component is not perfectly periodic. The minimum and maximum values at the end of December are larger than the minimum and maximum values at the start of January. Nonetheless, there is a clear reduction in component values in the winter months and increase in component values in the summer months.

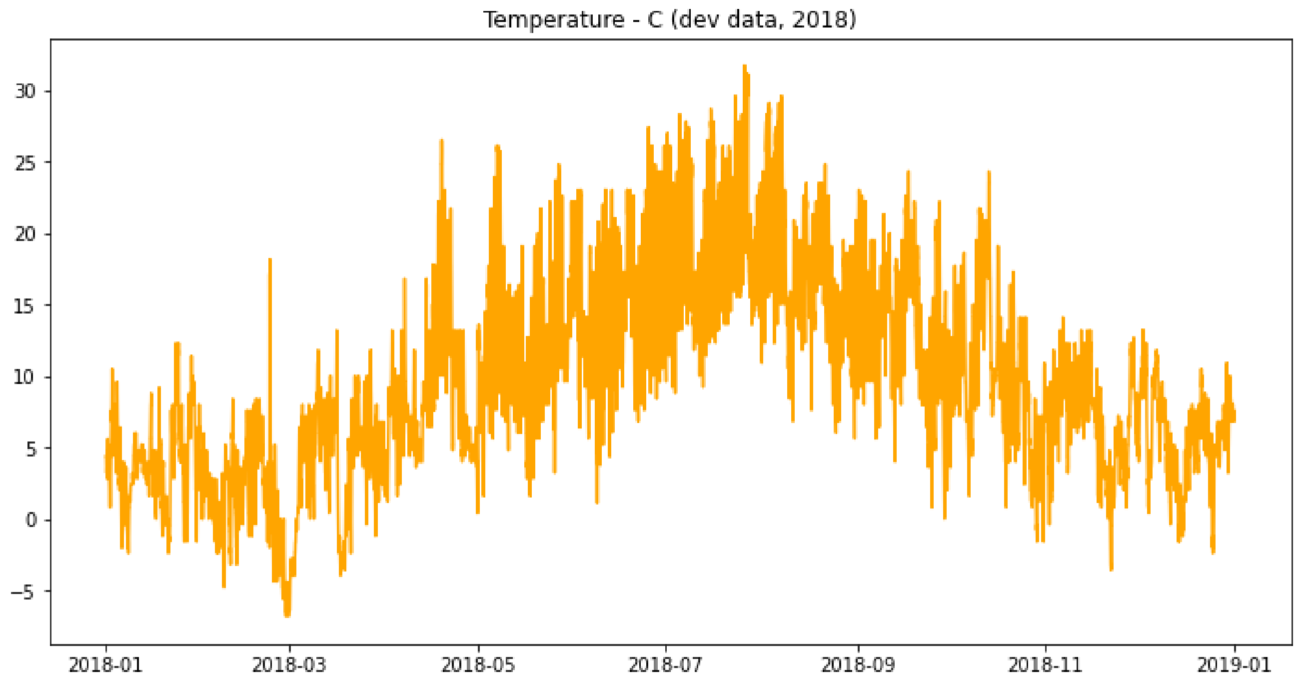
Split data¶

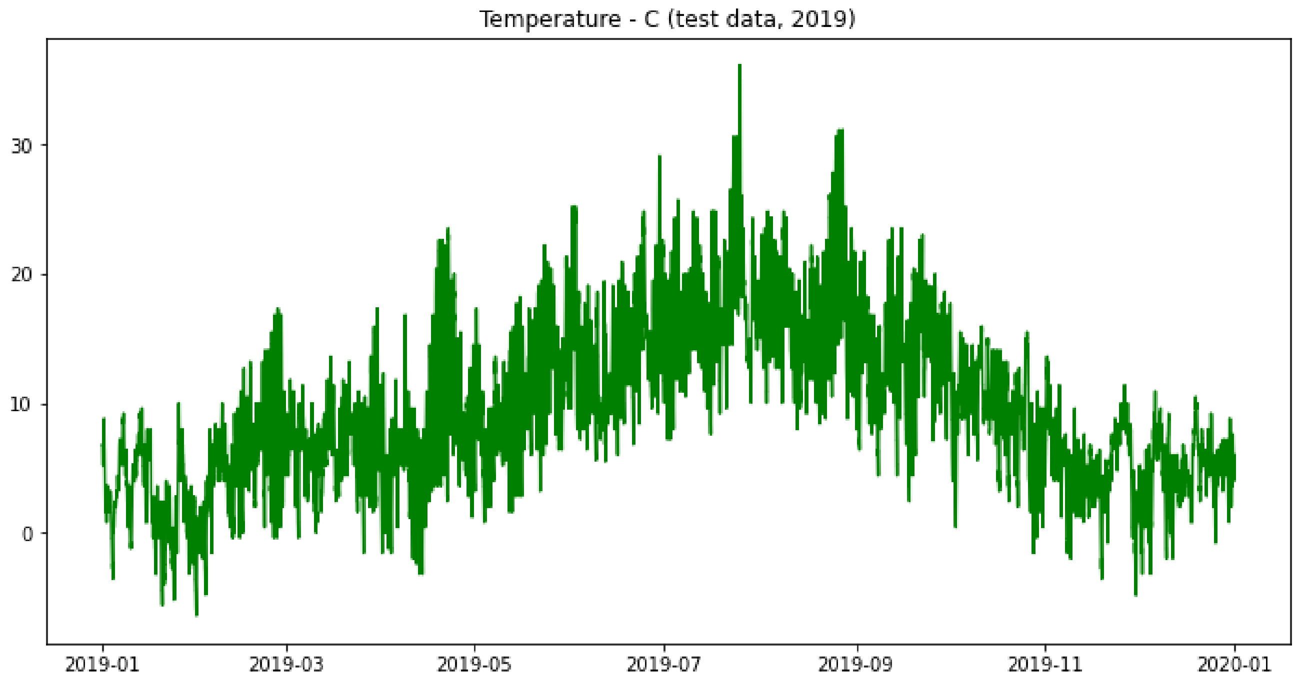
I use data from 2018 for validation, 2019 for testing and the remaining data for training. These are entirely arbitrary choices. This results in an approximate 84%, 8%, 8% split for the training, validation, and test sets respectively.

```
# 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
0
10
20
30
2008
2010
2012
2014
2016
2018
2020


Temperature - C (dev data, 2018)
−5
0
5
10
15
20
25
30
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
2019-01


Temperature - C (test data, 2019)
0
10
20
30
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01

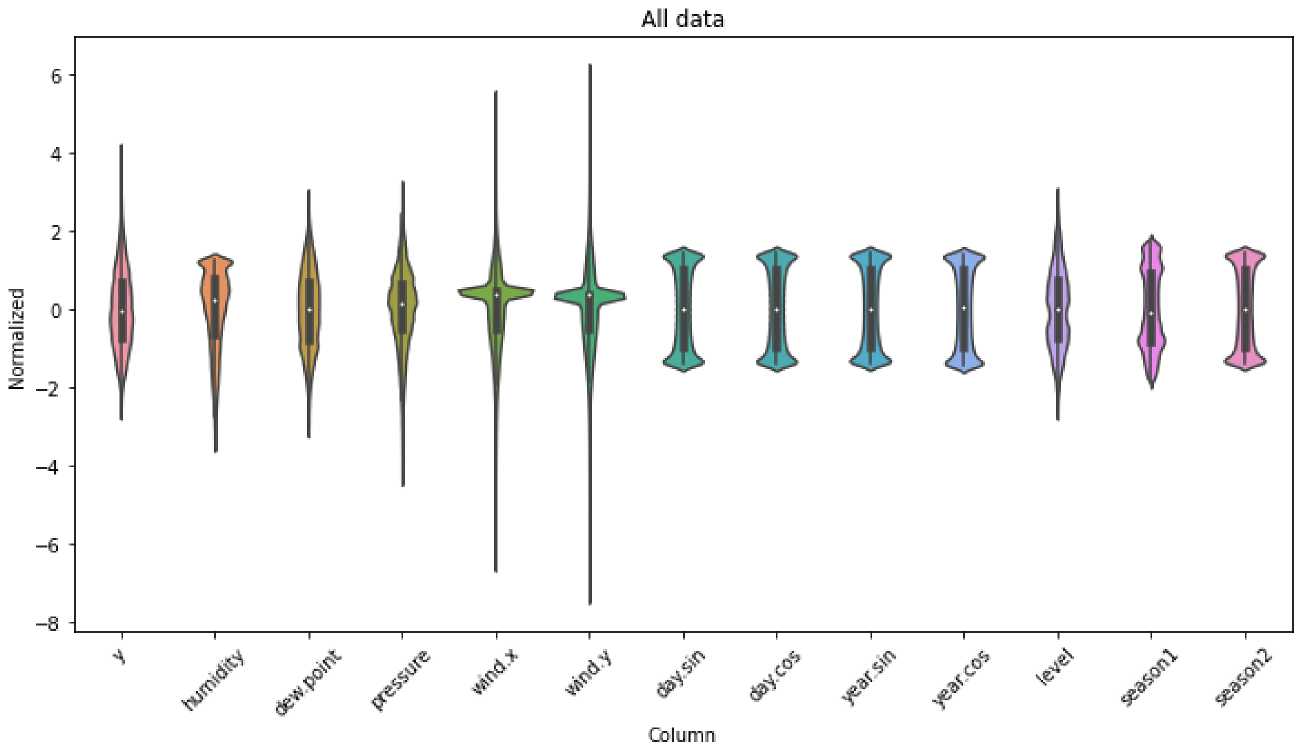
```
df.drop shape:  (223250, 13)
train shape:    (188210, 13)
valid shape:    (17520, 13)
test shape:     (17520, 13)
```

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

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 observations. 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. This is a fairly conservative starting point. It would be surprising if applying mixup between consecutive days of measurements didn't give better results. Applying mixup between inputs with equal temperature values will not improve performance and will increase run time.

I don't show it in this notebook, but adding this data augmentation makes a significant difference to loss values (for all model architectures considered). For example, here are results for a multi-layer perceptron (MLP) with 24 largs, 1 step ahead, 20 epochs 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)


    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


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

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

    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
```

LSTM Model Building

Long Short Term Memory networks, or LSTMs, were originally proposed in LONG SHORT TERM MEMORY. They are recurrent neural networks which have feedback connections.

LSTMs can take entire sequences of data as input and keep track of long-term dependencies. A LSTM unit is composed of a cell and three gates. The cell remembers values over arbitrary time intervals and the input, output and forget gates regulate the flow of information into and out of the cell.

TODO Include basic LSTM diagram

The following are a few points I consider when building these LSTM 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:

- Mixup
    - input mixup
    - time series mixup
- TBATS components
    - exponential smoothing state space model with Box-Cox Transformation, ARMA errors, Trend and Seasonal components
    - on multivariate data
- Time2Vec representation
    - Time2Vec: Learning a Vector Representation of Time
    - on univariate data
    - did not prove useful
    - as this notebook is getting quite long I've removed the Time2Vec work
        - still available in this commit
- VAR-style forecasts
    - Vector Auto-Regression forecasts for temperature, pressure, dew point and humidity
    - similar to statsmodels VAR baseline
    - did not prove useful
    - may or may not be worth considering VAR-style regression with multi-head output
        - that is, 4 output heads for: temperature, pressure, dew point and humidity
    - as this notebook is getting quite long I've removed the VAR-style forecasts work

        - still available in [this commit]
- Test time augmentation
    - uses data augmentation at the inference stage to improve forecasts
        - 5 forecasts were produced using mixup and then averaged
    - there was a marginal improvement
    - this may be worth trying again
    - as this notebook is getting quite long I've removed the test time augmentation work
        - still available in [this commit]

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 ...
- Mixup
    - factor - 1, 2, 3, 4, 5
        - run time increases with factor but gave some good results
        - as this notebook is getting quite long I've removed the mixup factor work
            - still available in [this commit]
    - alpha - 4 (recommended in [original publication])
    - time series mixup:
        - time diff - 1, ..., 48
            - period between 2 data subsets to run mixup on
            - increasing time diff roughly in line with lags gave some good results
            - as this notebook is getting quite long I've removed the mixup time diff work
                - still available in [this commit]
- Dropout and recurrent dropout
    - [dropout]
    - recurrent_dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state. Default: 0.
    - recurrent_dropout was very slow because it is unsupported in Nvidia's LSTM kernel
    - as this notebook is getting quite long I've removed the recurrent_dropout work
        - still available in [this commit]

- Epochs
    - training shows quite fast convergence so epochs is initially kept quite low (5 or 10)
    - final models are ran for 20 epochs

Model architectures to consider:

- Vanilla LSTM
    - single LSTM layer followed by Dense output layer
- Stacked LSTM
    - two LSTM layers
- Stacked bidirectional LSTM
    - two bidirectional LSTM layers
- ConvLSTM1D
    - LSTM layer where both input and recurrent transformations are convolutional

---

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:

```
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, sn
        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 = [0.0003, 10, 32, 5, 100, 25]  # 0.0003 too high
lrf_params = [0.000001, 10, 32, 5, 100, 25]
```

Next, define LSTM and other network architectures:

- build_vanilla_lstm_model
- build_stacked_lstm_model
- build_bidirectional_lstm_model
- build_convlstm1D_model

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

    return in_shape, out_shape


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

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

    if drop_out != 0.0:
      lstm.add(Dropout(drop_out))

    # Shape [batch, time, features] => [batch, feat_maps]
    lstm.add(LSTM(feat_maps,
                  return_sequences = False,
                  kernel_regularizer = regularizers.l2(kern_reg),
                  recurrent_regularizer = regularizers.l2(recu_reg)))

    if drop_out != 0.0:
      lstm.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      lstm.add(Dense(out_steps * out_feats,
                     kernel_constraint = maxnorm(3)))
    else:
      # Shape => [batch, out_steps * out_feats]
      lstm.add(Dense(out_steps * out_feats))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      lstm.add(Reshape([out_steps, out_feats]))
```

```
    return lstm


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

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

    # Shape [batch, time, features] => [batch, feat_maps]
    lstm.add(LSTM(feat_maps,
                  return_sequences = True,
                  kernel_regularizer = regularizers.l2(kern_reg),
                  recurrent_regularizer = regularizers.l2(recu_reg)))

    lstm.add(LSTM(feat_maps,
                  return_sequences = False,
                  kernel_regularizer = regularizers.l2(kern_reg),
                  recurrent_regularizer = regularizers.l2(recu_reg)))

    if drop_out != 0.0:
      lstm.add(Dropout(drop_out))

    lstm.add(Dense(feat_maps,
                   activation = 'relu',
                   kernel_regularizer = regularizers.l2(kern_reg)))

    lstm.add(Dense(int(feat_maps / 2),
                   activation = 'relu',
                   kernel_regularizer = regularizers.l2(kern_reg)))

    if drop_out != 0.0:
      #lstm.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      lstm.add(Dense(out_steps * out_feats,
                     kernel_constraint = maxnorm(3),
                     kernel_regularizer = regularizers.l2(kern_reg)))
    else:
      # Shape => [batch, out_steps * out_feats]
      lstm.add(Dense(out_steps * out_feats,
```

```
                 kernel_regularizer = regularizers.l2(kern_reg)))

    return lstm


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

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

    # Shape [batch, time, features] => [batch, feat_maps]
    lstm.add(Bidirectional(LSTM(feat_maps,
                                return_sequences = True,
                                kernel_regularizer = regularizers.l2(kern_reg),
                                recurrent_regularizer = regularizers.l2(recu_reg)

    lstm.add(Bidirectional(LSTM(feat_maps,
                                return_sequences = False,
                                kernel_regularizer = regularizers.l2(kern_reg),
                                recurrent_regularizer = regularizers.l2(recu_reg)

    lstm.add(Dense(feat_maps,
                   activation = 'relu',
                   kernel_regularizer = regularizers.l2(kern_reg)))

    lstm.add(Dense(int(feat_maps / 2),
                   activation = 'relu',
                   kernel_regularizer = regularizers.l2(kern_reg)))

    # Shape => [batch, out_steps]
    lstm.add(Dense(out_steps))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      lstm.add(Reshape([out_steps, out_feats]))

    return lstm


def build_conv1d_lstm_model(models, params):
```

```
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data)
    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']
    filters   = params['filters']
    kern_size = params['kern_size']

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

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

    if drop_out != 0.0:
      cnnlstm.add(Dropout(drop_out))

    cnnlstm.add(Conv1D(filters = filters,
                       activation = 'relu',
                       kernel_size = int(kern_size)))  #, input_shape=(n_timesteps
    cnnlstm.add(MaxPooling1D(pool_size = 2))

    # Shape [batch, time, features] => [batch, feat_maps]
    cnnlstm.add(LSTM(feat_maps,
                     return_sequences = False,
                     kernel_regularizer = regularizers.l2(kern_reg),
                     recurrent_regularizer = regularizers.l2(recu_reg)))

    cnnlstm.add(Dense(feat_maps,
                      activation = 'relu',
                      kernel_regularizer = regularizers.l2(kern_reg)))

    if drop_out != 0.0:
      cnnlstm.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      cnnlstm.add(Dense(out_steps * out_feats,
                        kernel_constraint = maxnorm(3)))
    else:
      cnnlstm.add(Dense(out_steps * out_feats))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      cnnlstm.add(Reshape([out_steps, out_feats]))

    return cnnlstm


def build_conv1d_dense_model(models, params):
```

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

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    filters   = params['filters']
    kern_size = params['kern_size']

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

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

    cnnlstm.add(Conv1D(filters = filters,
                       activation = 'relu',
                       kernel_size = int(kern_size)))  #, input_shape=(n_timesteps
    cnnlstm.add(MaxPooling1D(pool_size = 2))

    cnnlstm.add(Flatten())

    if drop_out != 0.0:
      cnnlstm.add(Dropout(drop_out))

    # Shape [batch, time, features] => [batch, feat_maps]
    cnnlstm.add(Dense(feat_maps,
                      activation = 'relu',
                      kernel_regularizer = regularizers.l2(kern_reg)))

    cnnlstm.add(Dense(int(feat_maps / 2),
                      activation = 'relu',
                      kernel_regularizer = regularizers.l2(kern_reg)))

    if drop_out != 0.0:
      cnnlstm.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      cnnlstm.add(Dense(out_steps * out_feats,
                        kernel_constraint = maxnorm(3)))
    else:
      cnnlstm.add(Dense(out_steps * out_feats))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      cnnlstm.add(Reshape([out_steps, out_feats]))

    return cnnlstm


def build_stacked_conv1d_lstm_model(models, params):
```

```
model_name = get_model_name(models, params)
data = models[model_name]['train']
in_shape, out_shape = get_io_shapes(data)
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']
filters   = params['filters']
kern_size = params['kern_size']

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

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

if drop_out != 0.0:
  cnnlstm.add(Dropout(drop_out))

cnnlstm.add(Conv1D(filters = filters,
                   kernel_size = kern_size,
                   activation = 'relu'))  #, input_shape=(n_timesteps,n_featu
cnnlstm.add(MaxPooling1D(pool_size = 2))

cnnlstm.add(Conv1D(filters = filters,
                   kernel_size = kern_size + 2,
                   activation = 'relu'))
cnnlstm.add(MaxPooling1D(pool_size = 2))

# Shape [batch, time, features] => [batch, feat_maps]
cnnlstm.add(LSTM(feat_maps,
                 return_sequences = True,
                 kernel_regularizer = regularizers.l2(kern_reg),
                 recurrent_regularizer = regularizers.l2(recu_reg)))

cnnlstm.add(LSTM(int(feat_maps / 2),
                 return_sequences = False,
                 kernel_regularizer = regularizers.l2(kern_reg),
                 recurrent_regularizer = regularizers.l2(recu_reg)))

if drop_out != 0.0:
  cnnlstm.add(Dropout(drop_out))
  # Shape => [batch, out_steps * out_feats]
  cnnlstm.add(Dense(out_steps * out_feats,
                    kernel_constraint = maxnorm(3)))
else:
  cnnlstm.add(Dense(out_steps * out_feats))

if len(out_shape) == 2:
  # Shape => [batch, out_steps, features].
```

```
      cnnlstm.add(Reshape([out_steps, out_feats]))

    return cnnlstm


def build_multihead_conv1d_lstm_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data)
    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']
    filters   = params['filters']
    kern_size = params['kern_size']

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

    # inputs
    inputs1 = Input(shape = in_shape)

    # head 1
    conv1 = Conv1D(filters = filters,
                   kernel_size = kern_size * 2 + 1,
                   activation = 'relu')(inputs1)
    drop1 = Dropout(drop_out)(conv1)
    pool1 = MaxPooling1D(pool_size = 2)(drop1)
    flat1 = Flatten()(pool1)

      # head 2
    conv2 = Conv1D(filters = filters,
                   kernel_size = kern_size * 3 + 1,
                   activation = 'relu')(inputs1)
    drop2 = Dropout(drop_out)(conv2)
    pool2 = MaxPooling1D(pool_size = 2)(drop2)
    flat2 = Flatten()(pool2)

      # head 3
    conv3 = Conv1D(filters = filters,
                   kernel_size = kern_size * 4 + 1,
                   activation = 'relu')(inputs1)
    drop3 = Dropout(drop_out)(conv3)
    pool3 = MaxPooling1D(pool_size = 2)(drop3)
    flat3 = Flatten()(pool3)

      # merge
    merged = concatenate([flat1, flat2, flat3])
    merged_r = Reshape((-1, 1))(merged)
```

```
      # interpretation
    lstm1 = LSTM(feat_maps,
                 return_sequences = False,
                 kernel_regularizer = regularizers.l2(kern_reg),
                 recurrent_regularizer = regularizers.l2(recu_reg))(merged_r)
    outputs = Dense(out_steps * out_feats)(lstm1)

    model = Model(inputs = inputs1, outputs = outputs, name = model_name)

    return model


def build_multihead_conv1d_dense_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    filters   = params['filters']
    kern_size = int(params['kern_size'])  # skopt tuple conversion probs

    #print("kern_size:", kern_size)

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

    # inputs
    inputs1 = Input(shape = in_shape)

    # head 1
    conv1 = Conv1D(filters = filters,
                   kernel_size = kern_size * 2 + 1,
                   activation = 'relu')(inputs1)
    drop1 = Dropout(drop_out)(conv1)
    pool1 = MaxPooling1D(pool_size = 2)(drop1)
    flat1 = Flatten()(pool1)

      # head 2
    conv2 = Conv1D(filters = filters,
                   kernel_size = kern_size * 3 + 1,
                   activation = 'relu')(inputs1)
    drop2 = Dropout(drop_out)(conv2)
    pool2 = MaxPooling1D(pool_size = 2)(drop2)
    flat2 = Flatten()(pool2)

      # head 3
    conv3 = Conv1D(filters = filters,
                   kernel_size = kern_size * 4 + 1,
                   activation = 'relu')(inputs1)
    drop3 = Dropout(drop_out)(conv3)
```

```
    pool3 = MaxPooling1D(pool_size = 2)(drop3)
    flat3 = Flatten()(pool3)

      # merge
    merged = concatenate([flat1, flat2, flat3])

      # interpretation
    if drop_out != 0.0:
      dense1  = Dense(feat_maps,
                      activation = 'relu',
                      kernel_constraint = maxnorm(3))(merged)
      dense2  = Dense(int(feat_maps / 2),
                      activation = 'relu',
                      kernel_constraint = maxnorm(3))(dense1)
      outputs = Dense(out_steps * out_feats,
                      kernel_constraint = maxnorm(3))(dense2)
    else:
      dense1  = Dense(feat_maps, activation = 'relu')(merged)
      dense2  = Dense(int(feat_maps / 2), activation = 'relu')(dense1)
      outputs = Dense(out_steps * out_feats)(dense2)

    model = Model(inputs = inputs1, outputs = outputs, name = model_name)

    return model


def build_conv2d_dense_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    filters   = params['filters']
    kern_size = params['kern_size']

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

    conv2ddense = Sequential(name = model_name)
    conv2ddense.add(InputLayer(input_shape = in_shape))

    if drop_out != 0.0:
      conv2ddense.add(Dropout(drop_out))

    conv2ddense.add(Reshape((in_shape[0], in_shape[1], 1)))

    conv2ddense.add(Conv2D(filters = filters,
                           kernel_size = (1, kern_size),
                           padding = 'same',
```

```
                         activation = 'relu'))  #, input_shape=(n_timesteps,n_fe
    conv2ddense.add(Flatten())
    #conv2ddense.add(MaxPooling2D(pool_size = (2, 2)))

    if drop_out != 0.0:
      conv2ddense.add(Dropout(drop_out))

    conv2ddense.add(Dense(feat_maps,
                          activation = 'relu',
                          kernel_regularizer = regularizers.l2(kern_reg)))
    conv2ddense.add(Dense(int(feat_maps / 2),
                          activation = 'relu',
                          kernel_regularizer = regularizers.l2(kern_reg)))

    if drop_out != 0.0:
      conv2ddense.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      conv2ddense.add(Dense(out_steps * out_feats,
                            kernel_constraint = maxnorm(3)))
    else:
      conv2ddense.add(Dense(out_steps * out_feats))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      conv2ddense.add(Reshape([out_steps, out_feats]))

    return conv2ddense


def build_convlstm1D_model(models, params):
    model_name = get_model_name(models, params)
    data = models[model_name]['train']
    in_shape, out_shape = get_io_shapes(data)
    out_steps = out_shape[0]

    feat_maps = params['feat_maps']
    drop_out  = params['drop_out']
    kern_reg  = params['kern_reg']
    filters   = params['filters']
    kern_size = params['kern_size']

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

    convlstm1D = Sequential(name = model_name)
    convlstm1D.add(InputLayer(input_shape = in_shape))

    if drop_out != 0.0:
      convlstm1D.add(Dropout(drop_out))

    convlstm1D.add(Reshape((in_shape[0], in_shape[1], 1)))  # worked but v slow
```

```
    convlstm1D.add(ConvLSTM1D(filters = filters,
                              kernel_size = kern_size,
                              data_format = 'channels_last'))  #,
    convlstm1D.add(Flatten())

    if drop_out != 0.0:
      convlstm1D.add(Dropout(drop_out))

    convlstm1D.add(Dense(feat_maps,
                         activation = 'relu',
                         kernel_regularizer = regularizers.l2(kern_reg)))
    convlstm1D.add(Dense(int(feat_maps / 2),
                         activation = 'relu',
                         kernel_regularizer = regularizers.l2(kern_reg)))

    if drop_out != 0.0:
      convlstm1D.add(Dropout(drop_out))
      # Shape => [batch, out_steps * out_feats]
      convlstm1D.add(Dense(out_steps * out_feats,
                           kernel_constraint = maxnorm(3)))
    else:
      convlstm1D.add(Dense(out_steps * out_feats))

    if len(out_shape) == 2:
      # Shape => [batch, out_steps, features].
      convlstm1D.add(Reshape([out_steps, out_feats]))

    return convlstm1D


def get_model(models, params):
    if params['model_type'] == 'lstm':
      model = build_vanilla_lstm_model(models, params)
    elif params['model_type'] == 's_lstm':
      model = build_stacked_lstm_model(models, params)
    elif params['model_type'] == 'b_lstm':
      model = build_bidirectional_lstm_model(models, params)
    elif params['model_type'] == 'conv1d_lstm':
      model = build_conv1d_lstm_model(models, params)
    elif params['model_type'] == 'conv1d_dense':
      model = build_conv1d_dense_model(models, params)
    elif params['model_type'] == 'conv2d_dense':
      model = build_conv2d_dense_model(models, params)
    elif params['model_type'] == 'conv_lstm1D':
      model = build_convlstm1D_model(models, params)
    elif params['model_type'] == 'mh_conv1d_lstm':
      model = build_multihead_conv1d_lstm_model(models, params)
    elif params['model_type'] == 'mh_conv1d_dense':
      model = build_multihead_conv1d_dense_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,
              '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'] == 'lstm':
      pass
    elif params['model_type'] == 's_lstm':
      pass
    elif params['model_type'] == 'b_lstm':
      pass
    elif params['model_type'] == 'conv1d_lstm':
      params.update({'lags': 144,
                     'bs':    32})
    elif params['model_type'] == 'conv1d_dense':
      params.update({'lags': 144,
                     'bs':    32})
    elif params['model_type'] == 'mh_conv1d_lstm':
      params.update({'lags': 144})
    elif params['model_type'] == 'mh_conv1d_dense':
      params.update({'lags': 144})
    elif params['model_type'] == 'conv2d_dense':
      params.update({'lags': 144})
    elif params['model_type'] == 'conv_lstm1D':
      params.update({'lags':      144,
                     'kern_size':   4,
                     'filters':    16})

    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
```

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() # Debugging

    # 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 examples of best near zero rmse forecasts
       Second row shows examples of worst positive rmse forecasts
```

```
    Third row shows examples of worst negative rmse forecasts

    Lagged observations are negative
    The day of the year the forecast begins on and the rmse value
    is shown on each subplot
  """

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


def mae(obs, preds):
    return np.median(np.abs(obs - preds))


def plot_horizon_metrics(models, model_name, dataset = 'valid'):
    """plot rmse and mae values for each individual step-ahead

    For a 48 step-ahead forecast rmse and mae values are plotted for
    each horizon value up to 48.
    """
```

```
# 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
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]  = 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()

axs[0].plot(range(1, horizon+1), rmse_h, label='LSTM')
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='LSTM')
    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', linesty
    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 vs 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]
    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]))
    test_long  = obs.reshape((obs.shape[0] * obs.shape[1]))

    # inverse transform using train mean & sd
    t_preds = inv_transform(scaler, preds_long, 'y', train_df.columns)
    t_obs   = inv_transform(scaler, test_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 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 main function here is `model_fitness_1s` which 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.

```
# !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_o

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 (arbitrarily) "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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: scikit-optimize in /usr/local/lib/python3.7/dis
Requirement already satisfied: pyaml>=16.9 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-
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.

skopt version: 0.9.0
```

Vanilla LSTM

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

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

- LSTM(return_sequences=True)
- Dense()

Finally, run vanilla LSTM models with optimised learning rates:

- 48 steps ahead
  - First feature selection
  - Second with mixup augmentation
  - Third optimise parameters
  - Fourth rerun best model(s) for more epochs and assess results

Feature selection

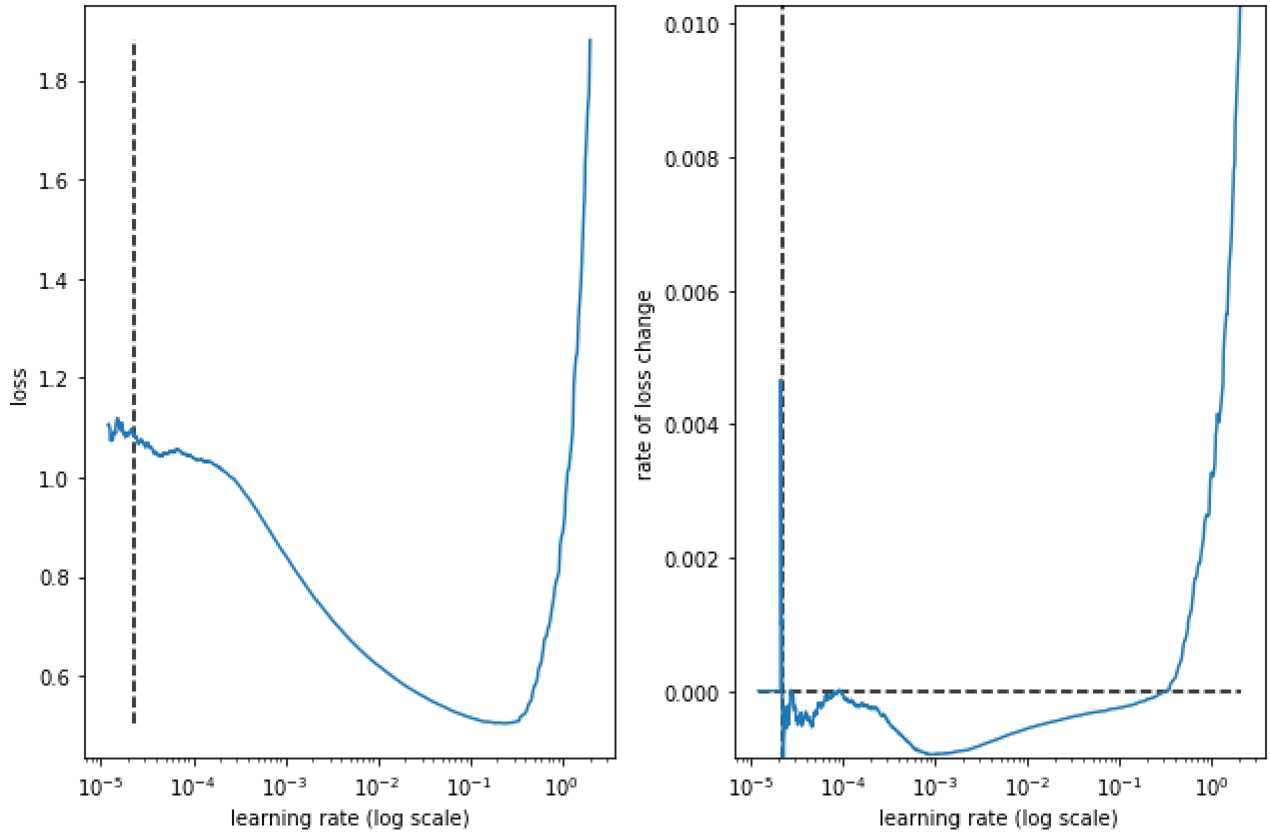
Compare performance from subsets of available variables.

Broadly speaking, I'm interested in comparing performance of sin/cos time components with TBATS time components. In hindsight, it would have been worthwhile also comparing one-hot encoded monthly variables.

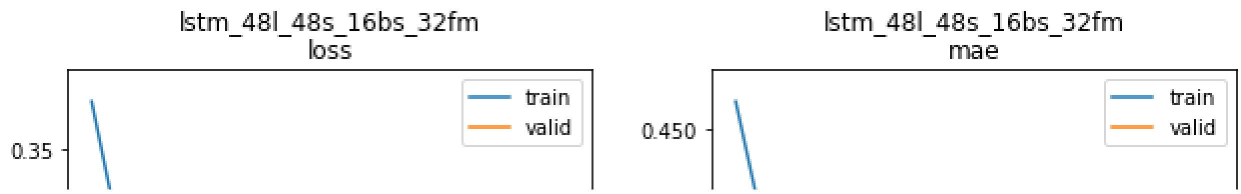
```
%%time


# def_cols     = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'da
y_col        = ['y']
```

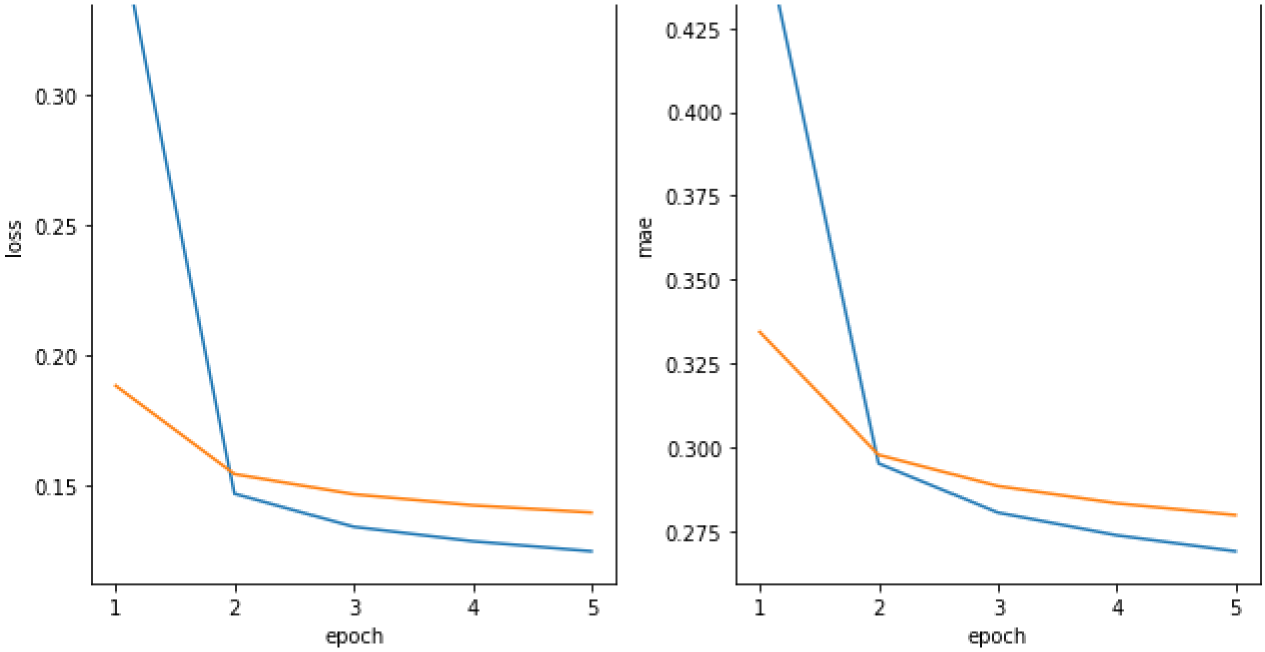
```
notime        = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y']
nowind        = ['y', 'humidity', 'dew.point', 'pressure', 'day.sin', 'day.cos',
var_cols      = ['y', 'humidity', 'dew.point', 'pressure']
day_col       = ['y', 'humidity', 'dew.point', 'pressure', 'day.sin']
year_col      = ['y', 'humidity', 'dew.point', 'pressure', 'year.sin']
tbats_cols    = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'le
tbats_day     = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'le
tbats_year    = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'le
tbats_nolevel = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'se

params = get_default_params('lstm')

sweep_values = {'xcols': [def_cols, y_col, notime, nowind, var_cols, day_col, year
models, xcol_model_names = sweep_param(models, params, sweep_values, verbose=True)

get_best_models(models, xcol_model_names)
get_best_models(models)

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

plot_perf_boxplot(models, 'val_loss', xcol_model_names)
plot_perf_boxplot(models, 'val_mae',  xcol_model_names)
```

```
xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'day.si
Epoch 1/5
11758/11758 [==============================] - 12s 910us/step - loss: 2.0560 -
```

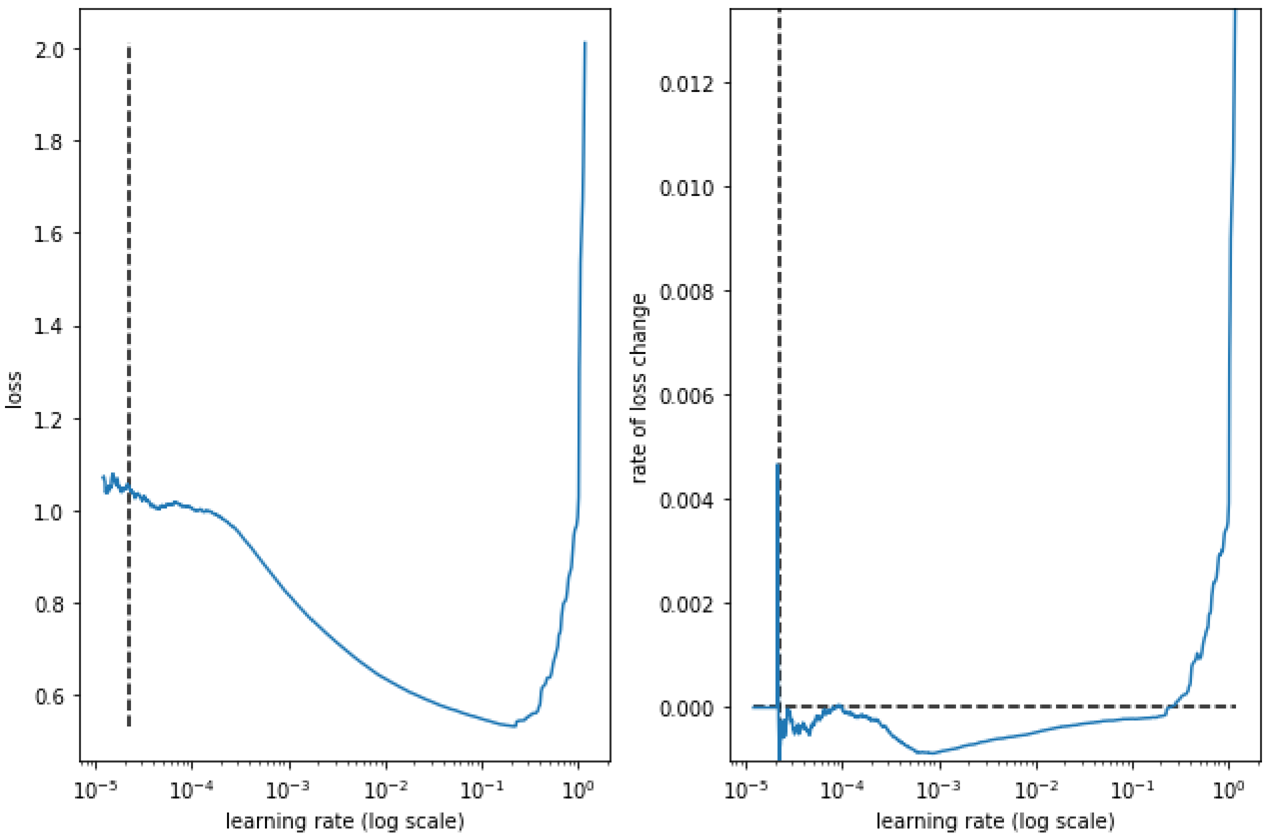



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_40 (LSTM)              (None, 32)                5504

 dense_577 (Dense)           (None, 48)                1584

 reshape_101 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 59s - loss: 0.3681 - mae: 0.4582 - val_loss: 0.1882 - val_mae: 0
Epoch 2/5
11758/11758 - 57s - loss: 0.1468 - mae: 0.2951 - val_loss: 0.1543 - val_mae: 0
Epoch 3/5
11758/11758 - 57s - loss: 0.1341 - mae: 0.2804 - val_loss: 0.1465 - val_mae: 0
Epoch 4/5
11758/11758 - 57s - loss: 0.1285 - mae: 0.2736 - val_loss: 0.1423 - val_mae: 0
Epoch 5/5
11758/11758 - 58s - loss: 0.1247 - mae: 0.2689 - val_loss: 0.1395 - val_mae: 0
```



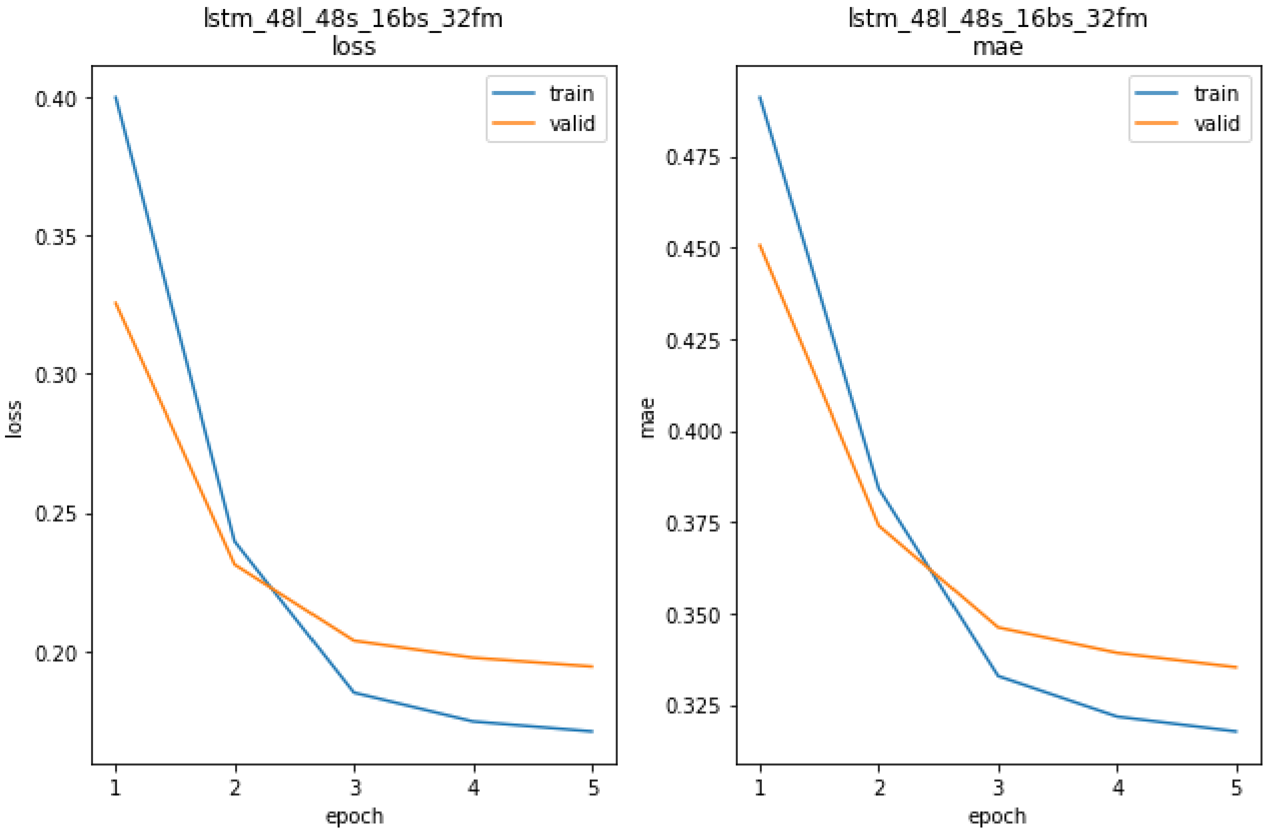



```
lstm_48l_48s_16bs_32fm train min loss: 0.124658 mae: 0.268870   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.139528 mae: 0.279673   epoch: 5

xcols : ['y']
Epoch 1/5
11758/11758 [==============================] - 12s 873us/step - loss: 2.1626 -
```

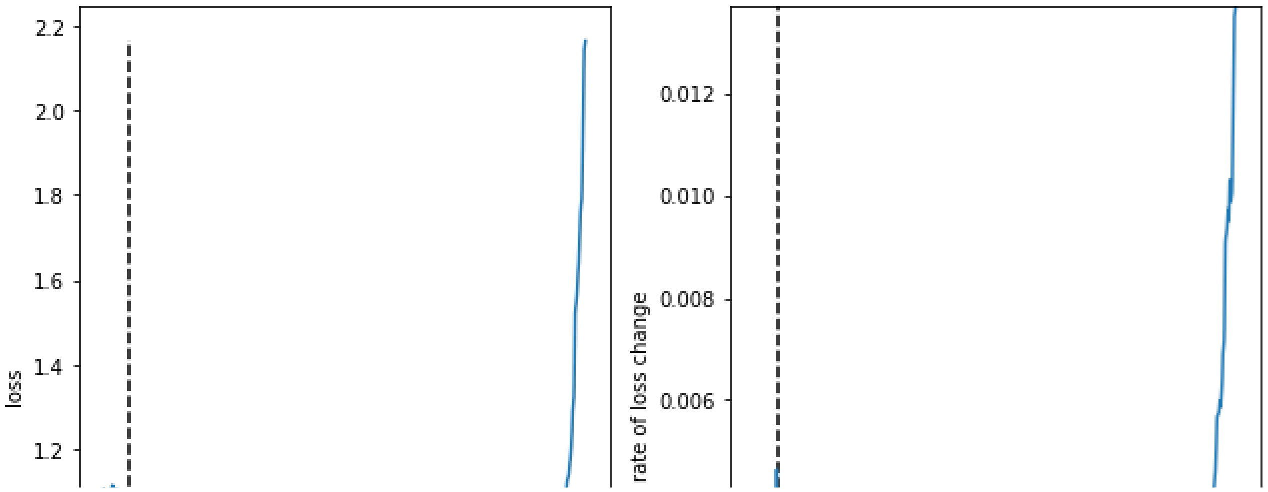


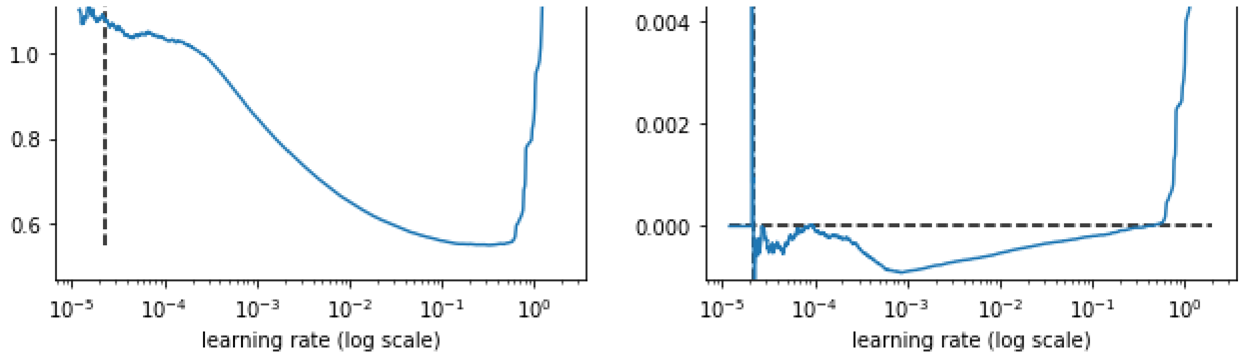

```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_41 (LSTM)              (None, 32)                4352

 dense_578 (Dense)           (None, 48)                1584

 reshape_102 (Reshape)       (None, 48, 1)             0
```

```
=================================================================
Total params: 5,936
Trainable params: 5,936
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 58s - loss: 0.3999 - mae: 0.4911 - val_loss: 0.3256 - val_mae: 0
Epoch 2/5
11758/11758 - 57s - loss: 0.2397 - mae: 0.3841 - val_loss: 0.2313 - val_mae: 0
Epoch 3/5
11758/11758 - 56s - loss: 0.1850 - mae: 0.3328 - val_loss: 0.2038 - val_mae: 0
Epoch 4/5
11758/11758 - 56s - loss: 0.1746 - mae: 0.3217 - val_loss: 0.1977 - val_mae: 0
Epoch 5/5
11758/11758 - 56s - loss: 0.1710 - mae: 0.3177 - val_loss: 0.1945 - val_mae: 0
```

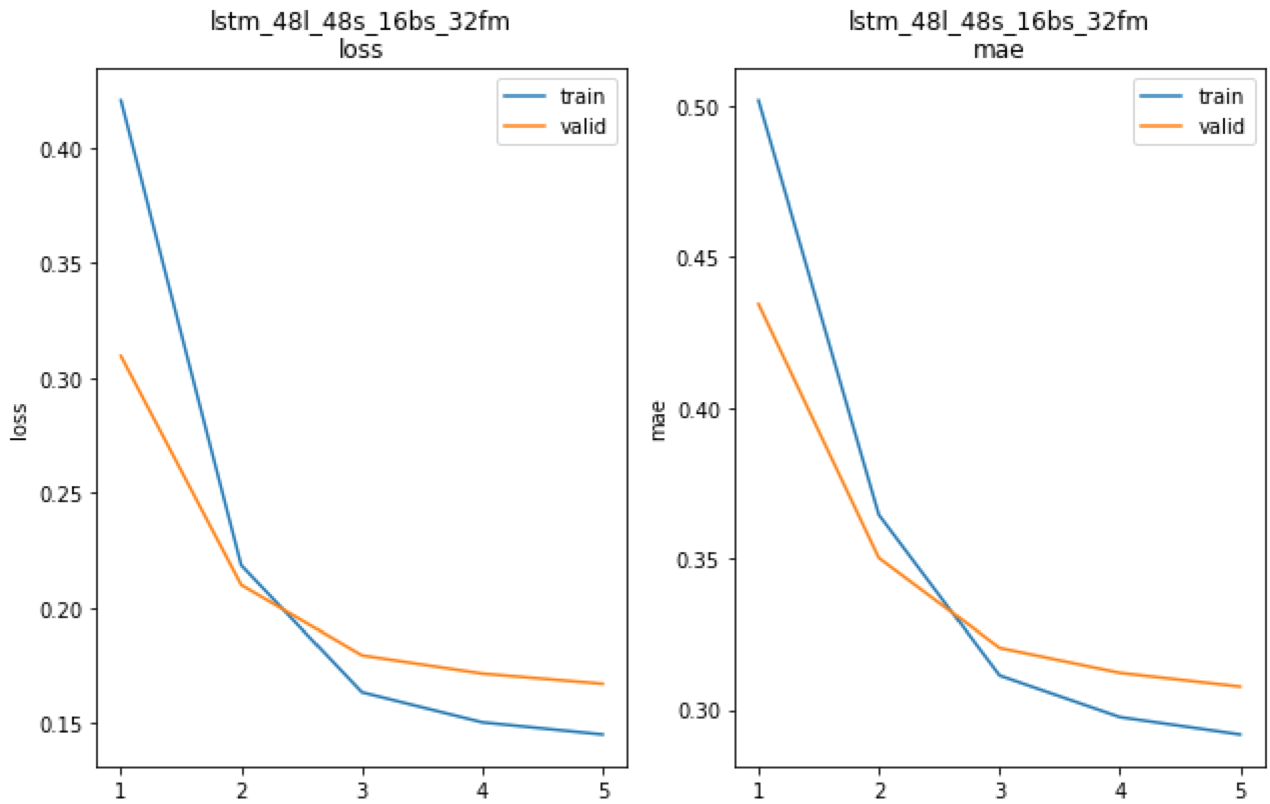



```
lstm_48l_48s_16bs_32fm train min loss: 0.171015 mae: 0.317662   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.194473 mae: 0.335241   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y']
Epoch 1/5
11758/11758 [==============================] - 13s 965us/step - loss: 2.2179 -
```



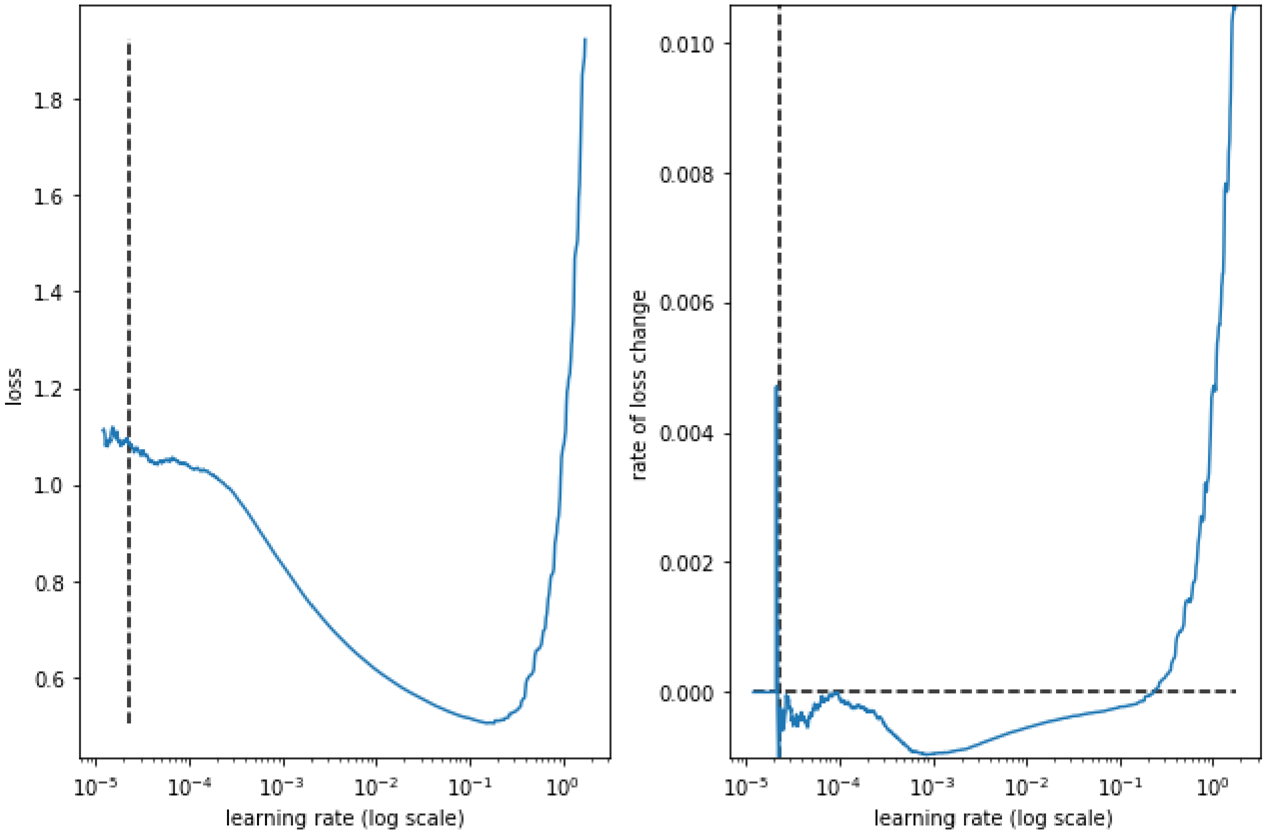



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_42 (LSTM)              (None, 32)                4992

 dense_579 (Dense)           (None, 48)                1584

 reshape_103 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,576
Trainable params: 6,576
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 58s - loss: 0.4206 - mae: 0.5018 - val_loss: 0.3096 - val_mae: 0
Epoch 2/5
11758/11758 - 57s - loss: 0.2186 - mae: 0.3646 - val_loss: 0.2100 - val_mae: 0
Epoch 3/5
11758/11758 - 57s - loss: 0.1635 - mae: 0.3114 - val_loss: 0.1796 - val_mae: 0
Epoch 4/5
11758/11758 - 57s - loss: 0.1505 - mae: 0.2976 - val_loss: 0.1717 - val_mae: 0
Epoch 5/5
11758/11758 - 57s - loss: 0.1452 - mae: 0.2919 - val_loss: 0.1673 - val_mae: 0
```



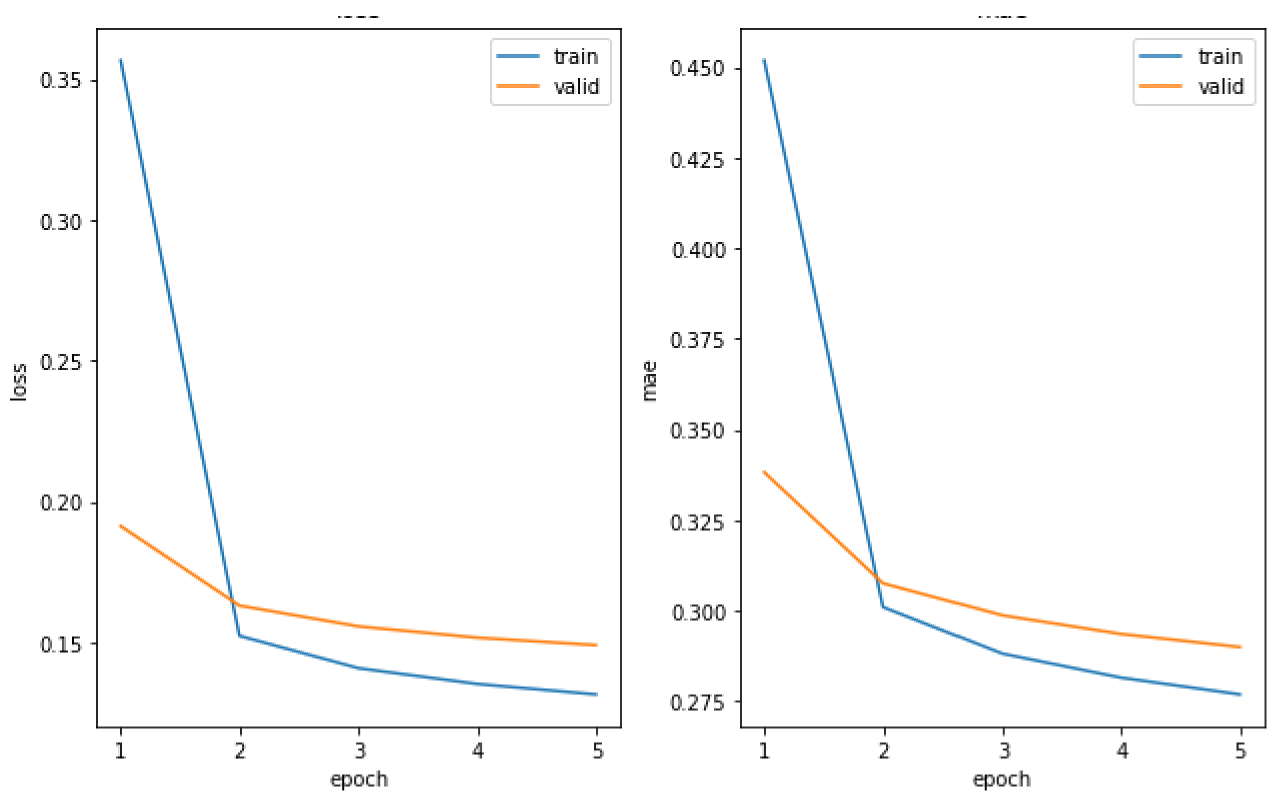
epoch epoch

```
lstm_48l_48s_16bs_32fm train min loss: 0.145205 mae: 0.291862   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.167261 mae: 0.307733   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'day.sin', 'day.cos', 'year
Epoch 1/5
11758/11758 [==============================] - 13s 955us/step - loss: 2.0452 -
```

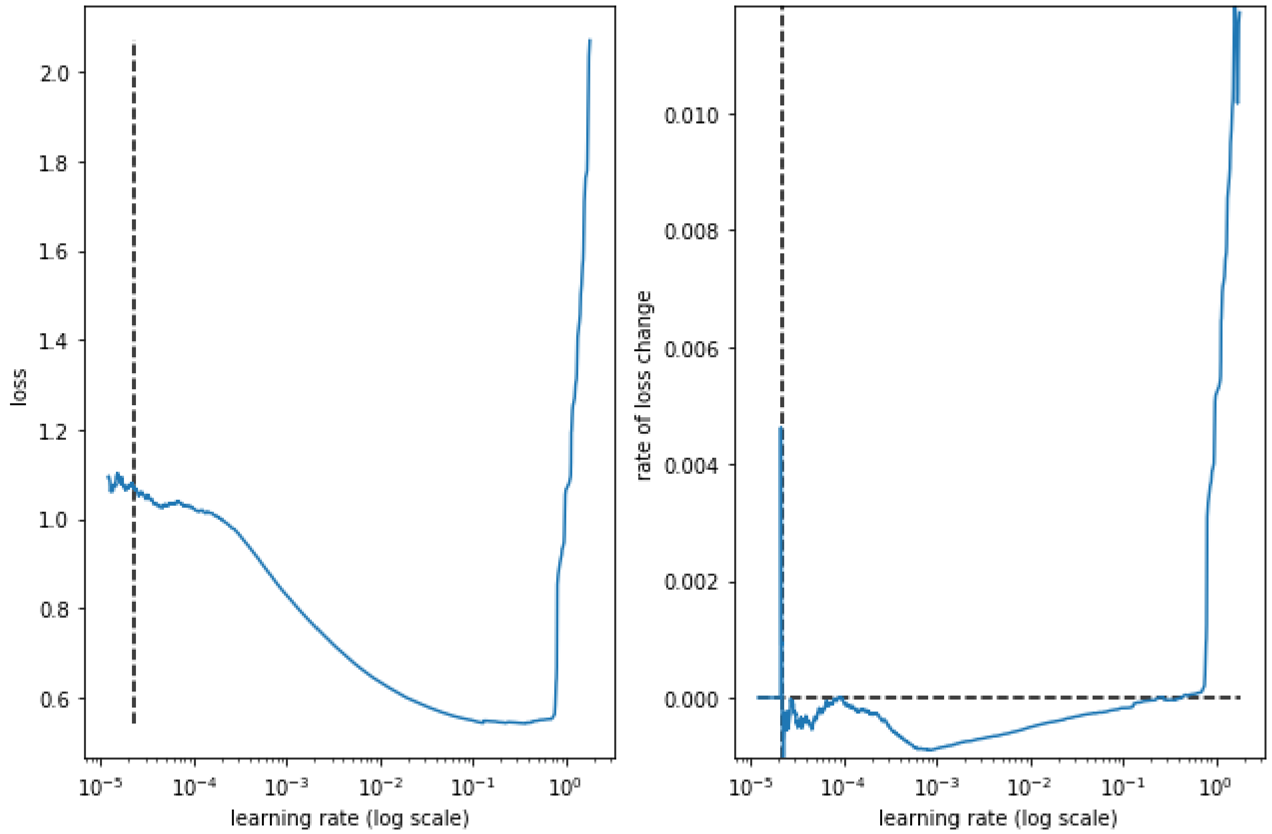



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_43 (LSTM)              (None, 32)                5248

 dense_580 (Dense)           (None, 48)                1584

 reshape_104 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,832
Trainable params: 6,832
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 59s - loss: 0.3566 - mae: 0.4519 - val_loss: 0.1913 - val_mae: 0
Epoch 2/5
11758/11758 - 58s - loss: 0.1523 - mae: 0.3009 - val_loss: 0.1630 - val_mae: 0
Epoch 3/5
11758/11758 - 56s - loss: 0.1408 - mae: 0.2880 - val_loss: 0.1557 - val_mae: 0
Epoch 4/5
11758/11758 - 57s - loss: 0.1352 - mae: 0.2814 - val_loss: 0.1516 - val_mae: 0
Epoch 5/5
11758/11758 - 56s - loss: 0.1315 - mae: 0.2768 - val_loss: 0.1490 - val_mae: 0
```

lstm_48l_48s_16bs_32fm loss lstm_48l_48s_16bs_32fm mae

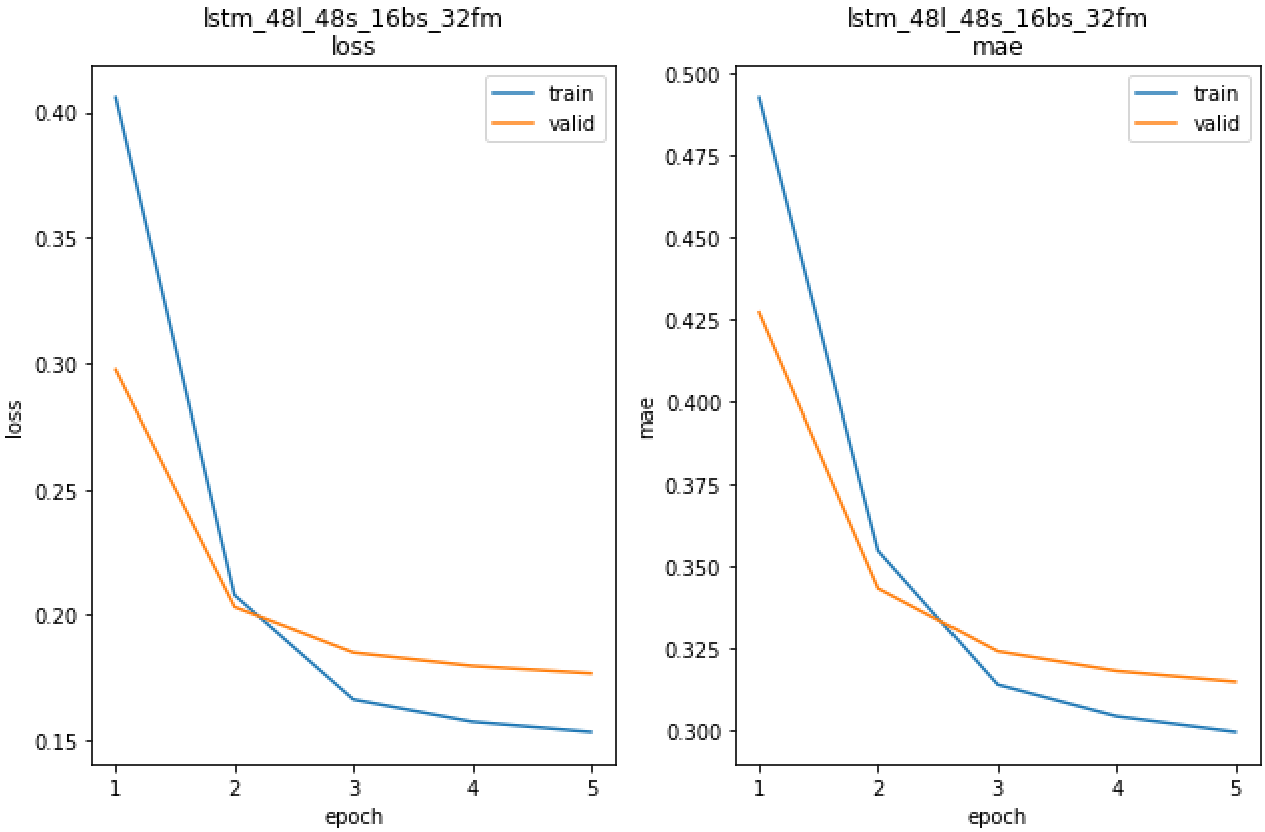



```
lstm_48l_48s_16bs_32fm train min loss: 0.131468 mae: 0.276754   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.148983 mae: 0.289858   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure']
Epoch 1/5
11758/11758 [==============================] - 13s 962us/step - loss: 2.1937 -
```

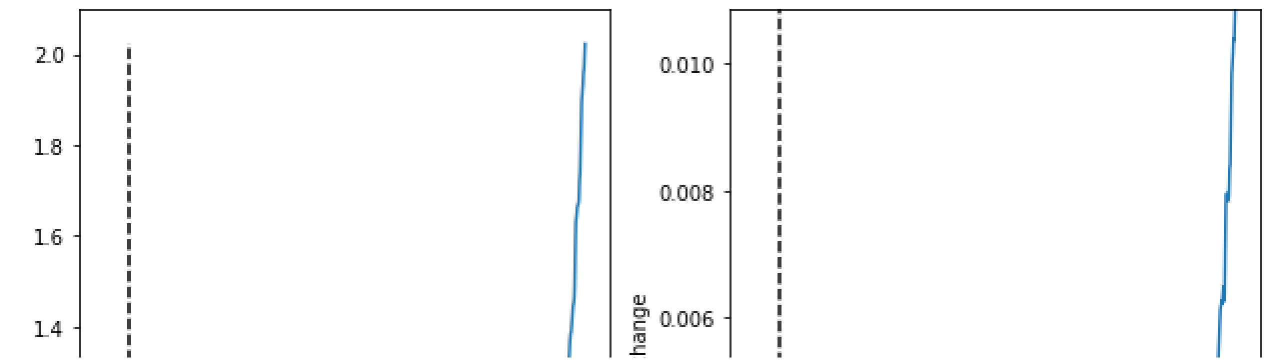


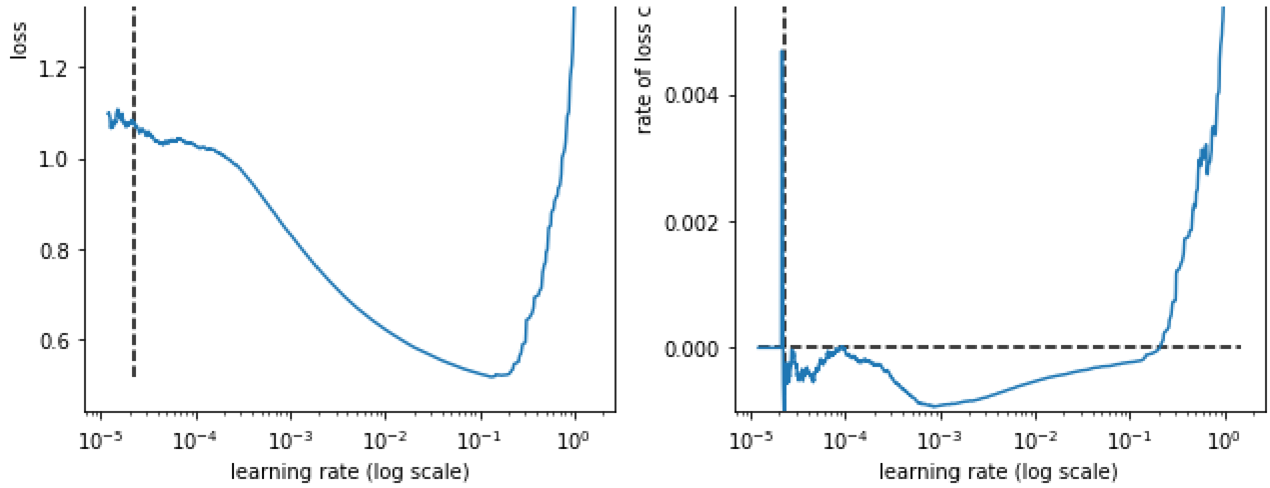

```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_44 (LSTM)              (None, 32)                4736
```

```
 dense_581 (Dense)          (None, 48)               1584

 reshape_105 (Reshape)      (None, 48, 1)            0

=================================================================
Total params: 6,320
Trainable params: 6,320
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 58s - loss: 0.4058 - mae: 0.4926 - val_loss: 0.2974 - val_mae: 0
Epoch 2/5
11758/11758 - 57s - loss: 0.2079 - mae: 0.3548 - val_loss: 0.2032 - val_mae: 0
Epoch 3/5
11758/11758 - 56s - loss: 0.1663 - mae: 0.3139 - val_loss: 0.1850 - val_mae: 0
Epoch 4/5
11758/11758 - 56s - loss: 0.1574 - mae: 0.3043 - val_loss: 0.1797 - val_mae: 0
Epoch 5/5
11758/11758 - 57s - loss: 0.1534 - mae: 0.2995 - val_loss: 0.1767 - val_mae: 0
```

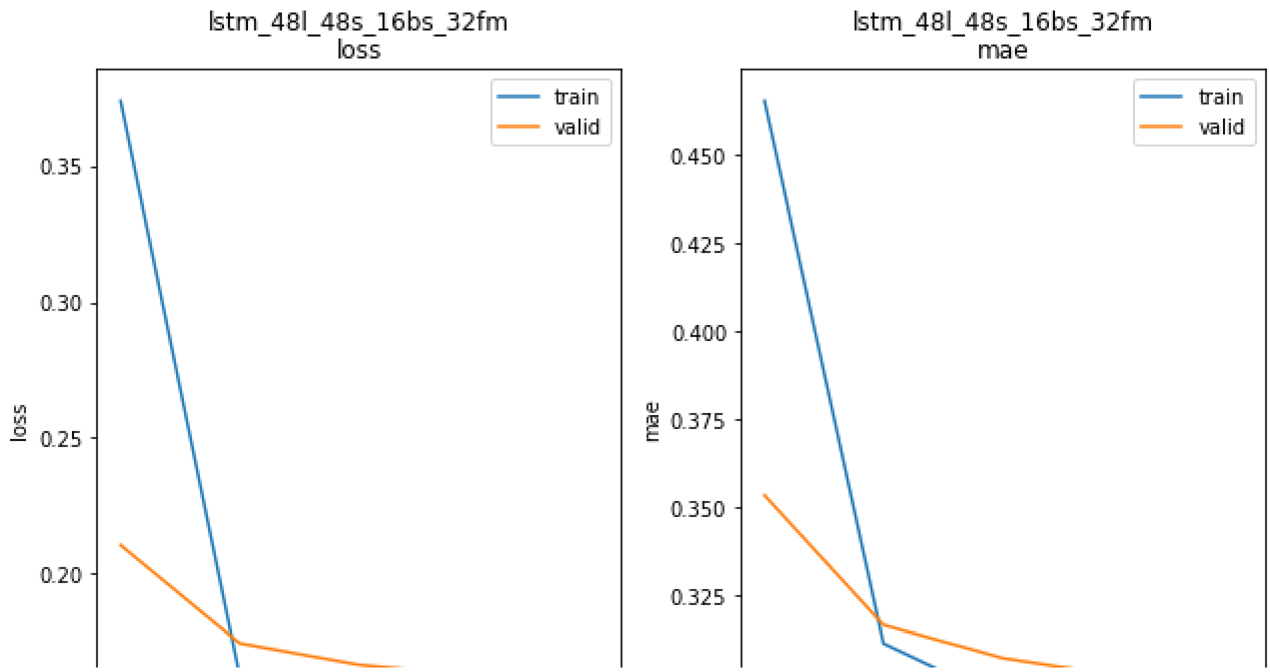


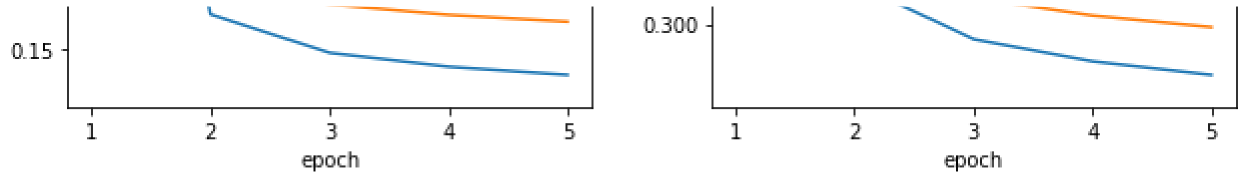

```
lstm_48l_48s_16bs_32fm train min loss: 0.153370 mae: 0.299531   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.176722 mae: 0.314810   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'day.sin']
Epoch 1/5
11758/11758 [==============================] - 13s 961us/step - loss: 2.0918 -
```



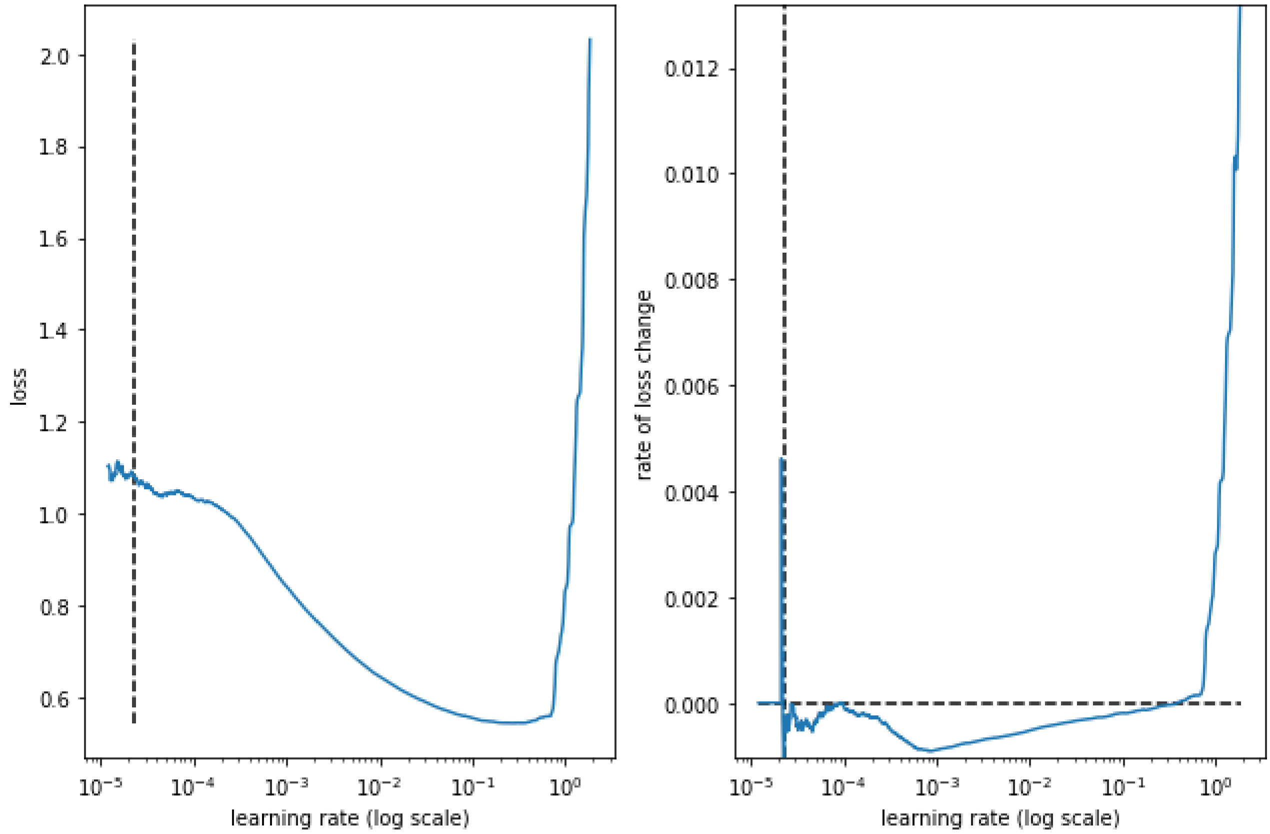



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_45 (LSTM)              (None, 32)                4864

 dense_582 (Dense)           (None, 48)                1584

 reshape_106 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,448
Trainable params: 6,448
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 58s - loss: 0.3739 - mae: 0.4651 - val_loss: 0.2105 - val_mae: 0
Epoch 2/5
11758/11758 - 58s - loss: 0.1629 - mae: 0.3116 - val_loss: 0.1745 - val_mae: 0
Epoch 3/5
11758/11758 - 57s - loss: 0.1488 - mae: 0.2957 - val_loss: 0.1666 - val_mae: 0
Epoch 4/5
11758/11758 - 58s - loss: 0.1436 - mae: 0.2895 - val_loss: 0.1627 - val_mae: 0
Epoch 5/5
11758/11758 - 58s - loss: 0.1406 - mae: 0.2856 - val_loss: 0.1602 - val_mae: 0
```



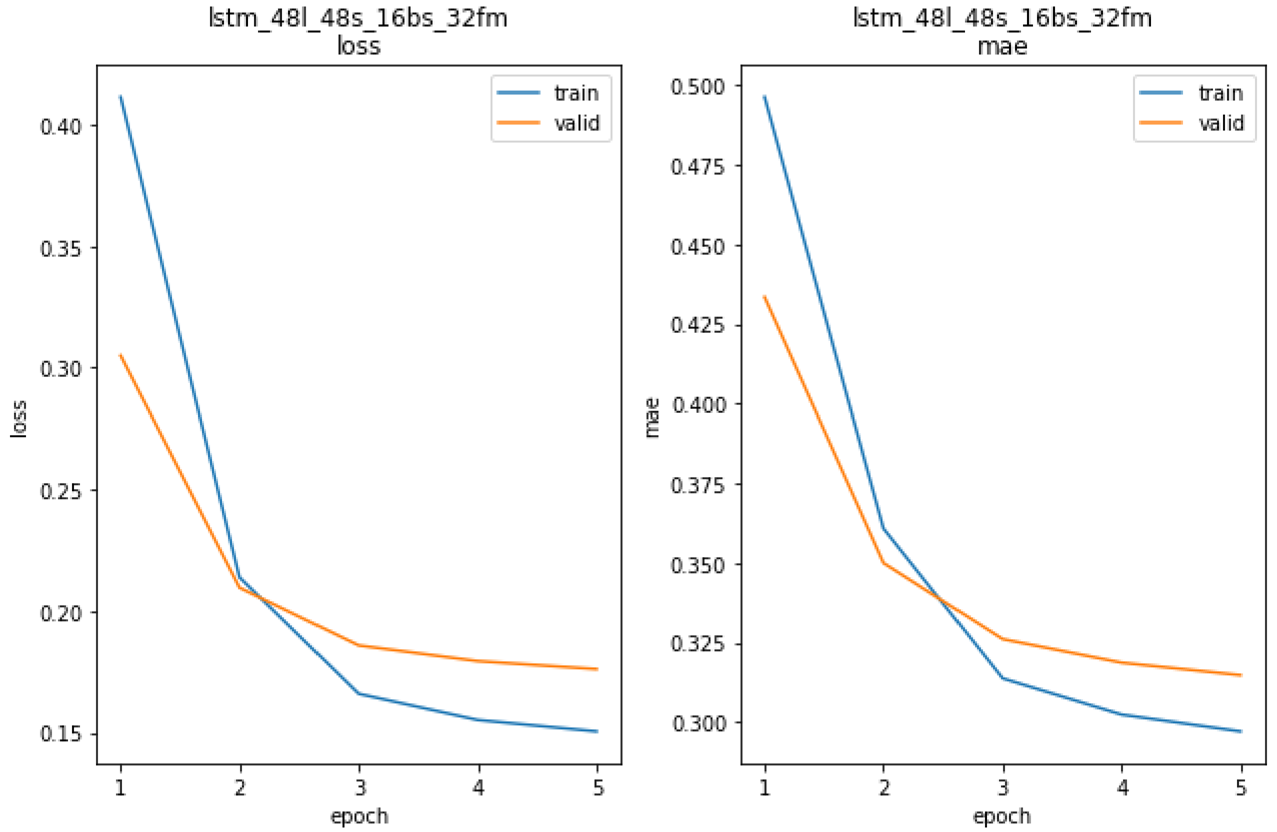



```
lstm_48l_48s_16bs_32fm train min loss: 0.140612 mae: 0.285630   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.160221 mae: 0.299164   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'year.sin']
Epoch 1/5
11758/11758 [==============================] - 13s 1ms/step - loss: 2.2162 - m
```

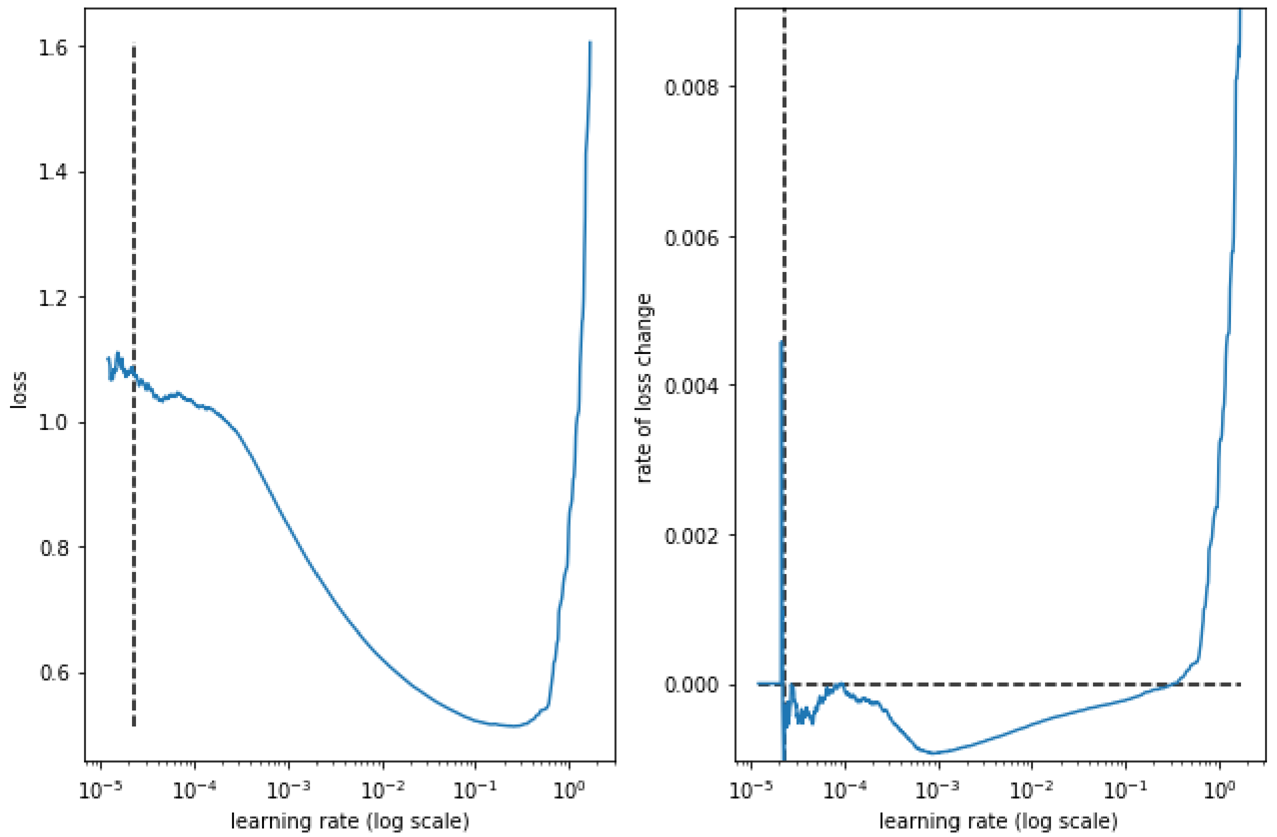



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_46 (LSTM)              (None, 32)                4864

 dense_583 (Dense)           (None, 48)                1584

 reshape_107 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,448
Trainable params: 6,448
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 61s - loss: 0.4114 - mae: 0.4963 - val_loss: 0.3050 - val_mae: 0
Epoch 2/5
11758/11758 - 58s - loss: 0.2139 - mae: 0.3609 - val_loss: 0.2096 - val_mae: 0
Epoch 3/5
11758/11758 - 58s - loss: 0.1659 - mae: 0.3138 - val_loss: 0.1858 - val_mae: 0
Epoch 4/5
11758/11758 - 59s - loss: 0.1552 - mae: 0.3024 - val_loss: 0.1794 - val_mae: 0
```

```
Epoch 5/5
11758/11758 - 58s - loss: 0.1505 - mae: 0.2971 - val_loss: 0.1761 - val_mae: 0
```

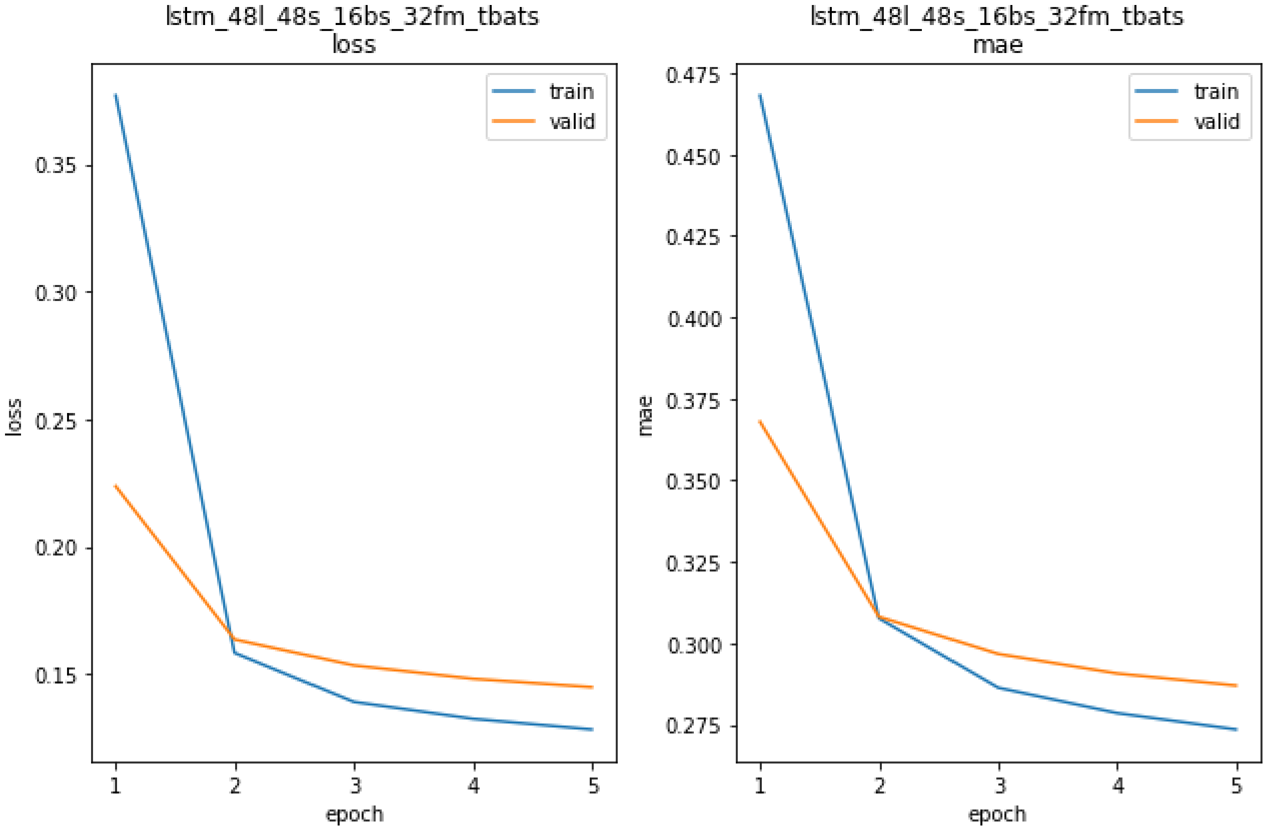



```
lstm_48l_48s_16bs_32fm train min loss: 0.150469 mae: 0.297086   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.176118 mae: 0.314774   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'level'
Epoch 1/5
11758/11758 [==============================] - 14s 1ms/step - loss: 2.0488 - m
```

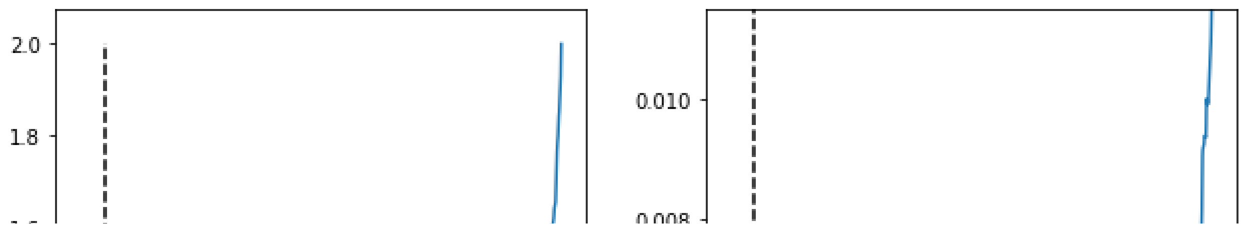


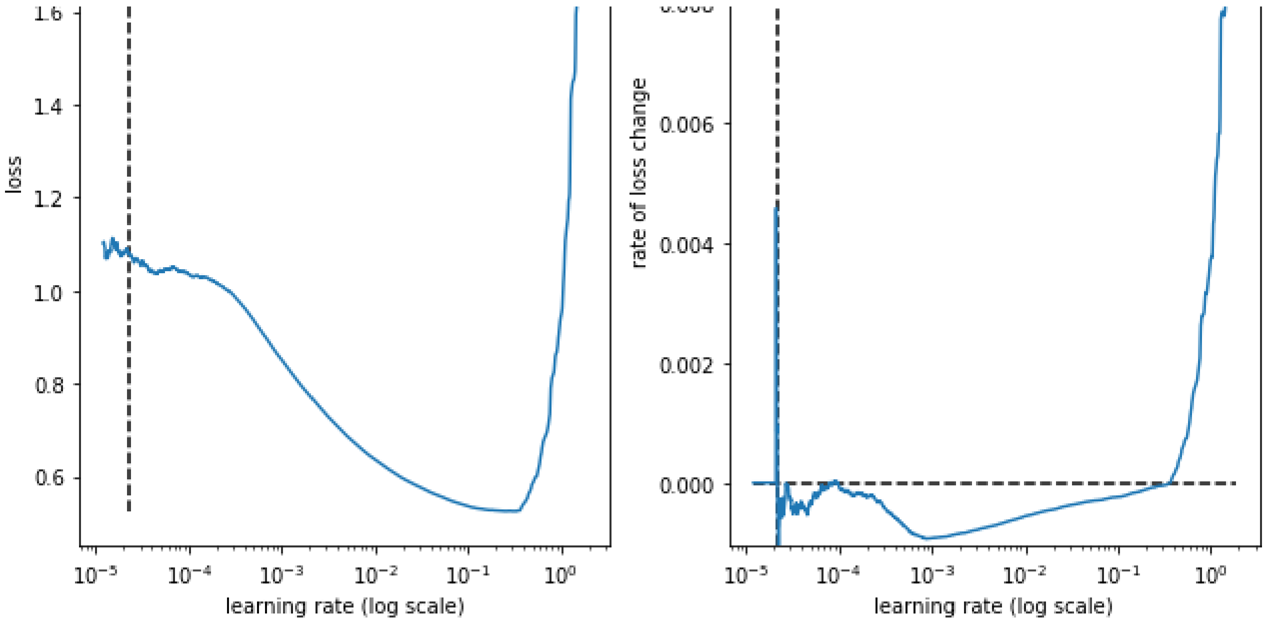

```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm_tbats"
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_47 (LSTM)               (None, 32)                5376

dense_584 (Dense)            (None, 48)                1584

reshape_108 (Reshape)        (None, 48, 1)             0

=================================================================
Total params: 6,960
Trainable params: 6,960
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 60s - loss: 0.3770 - mae: 0.4682 - val_loss: 0.2238 - val_mae: (
Epoch 2/5
11758/11758 - 56s - loss: 0.1583 - mae: 0.3077 - val_loss: 0.1636 - val_mae: (
Epoch 3/5
11758/11758 - 57s - loss: 0.1391 - mae: 0.2863 - val_loss: 0.1534 - val_mae: (
Epoch 4/5
11758/11758 - 56s - loss: 0.1324 - mae: 0.2785 - val_loss: 0.1481 - val_mae: (
Epoch 5/5
11758/11758 - 56s - loss: 0.1283 - mae: 0.2735 - val_loss: 0.1449 - val_mae: (
```

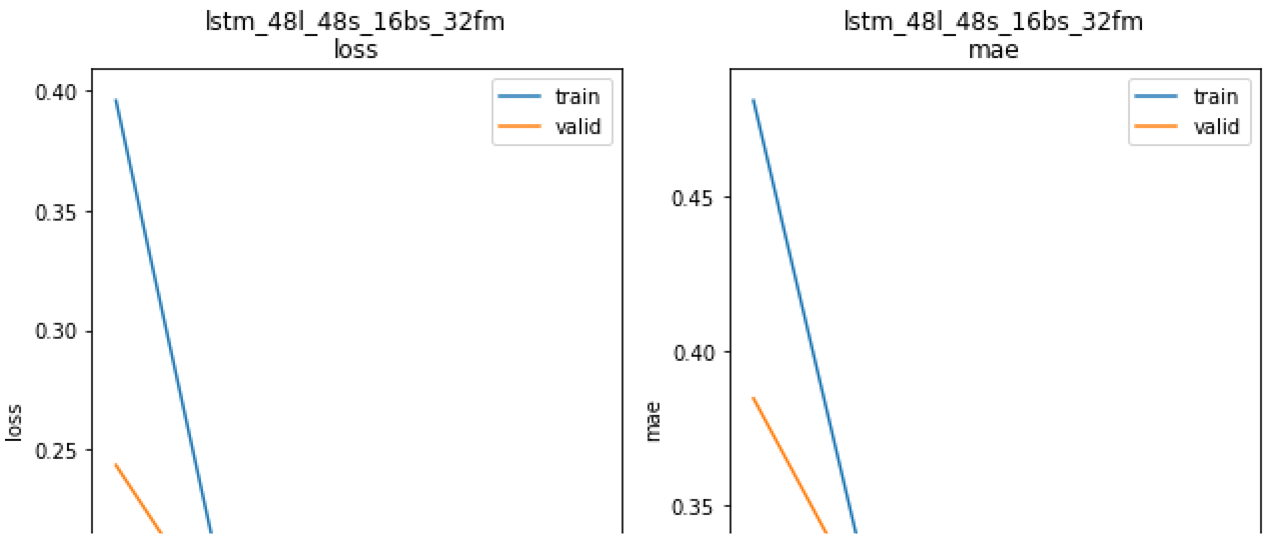


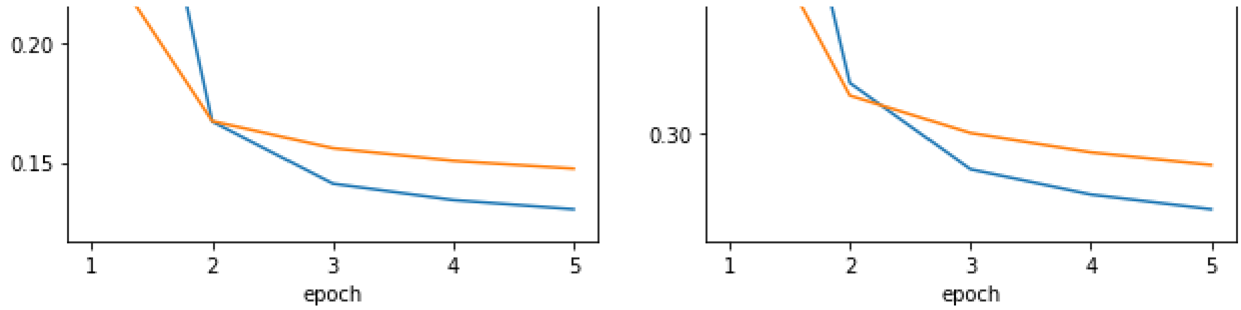

```
lstm_48l_48s_16bs_32fm_tbats train min loss: 0.128260   mae: 0.273506   epoch:
lstm_48l_48s_16bs_32fm_tbats valid min loss: 0.144865   mae: 0.286991   epoch:

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'level'
Epoch 1/5
11758/11758 [==============================] - 14s 1ms/step - loss: 2.1206 - m
```



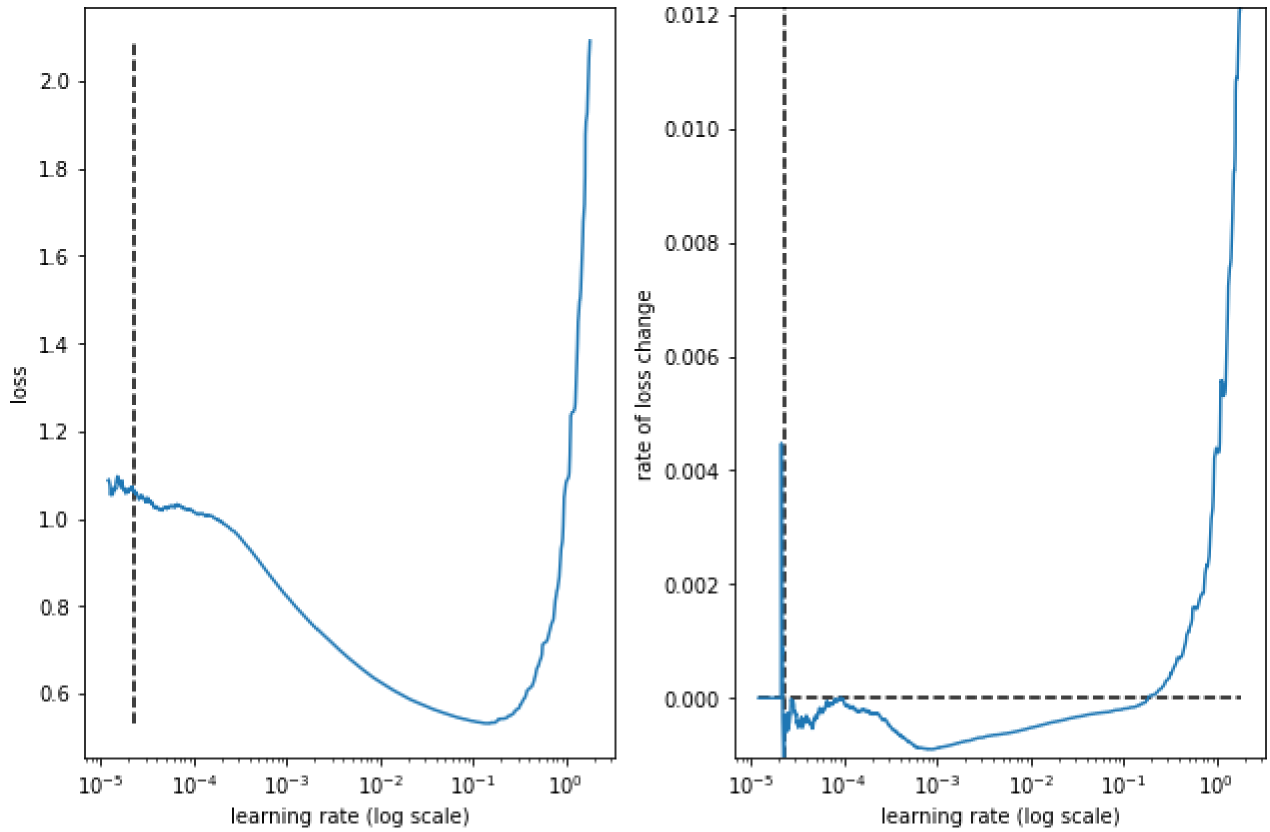



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_48 (LSTM)              (None, 32)                5248

 dense_585 (Dense)           (None, 48)                1584

 reshape_109 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,832
Trainable params: 6,832
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 60s - loss: 0.3960 - mae: 0.4813 - val_loss: 0.2434 - val_mae: (
Epoch 2/5
11758/11758 - 58s - loss: 0.1671 - mae: 0.3166 - val_loss: 0.1673 - val_mae: (
Epoch 3/5
11758/11758 - 58s - loss: 0.1411 - mae: 0.2886 - val_loss: 0.1559 - val_mae: (
Epoch 4/5
11758/11758 - 58s - loss: 0.1342 - mae: 0.2805 - val_loss: 0.1506 - val_mae: (
Epoch 5/5
11758/11758 - 59s - loss: 0.1303 - mae: 0.2757 - val_loss: 0.1474 - val_mae: (
```



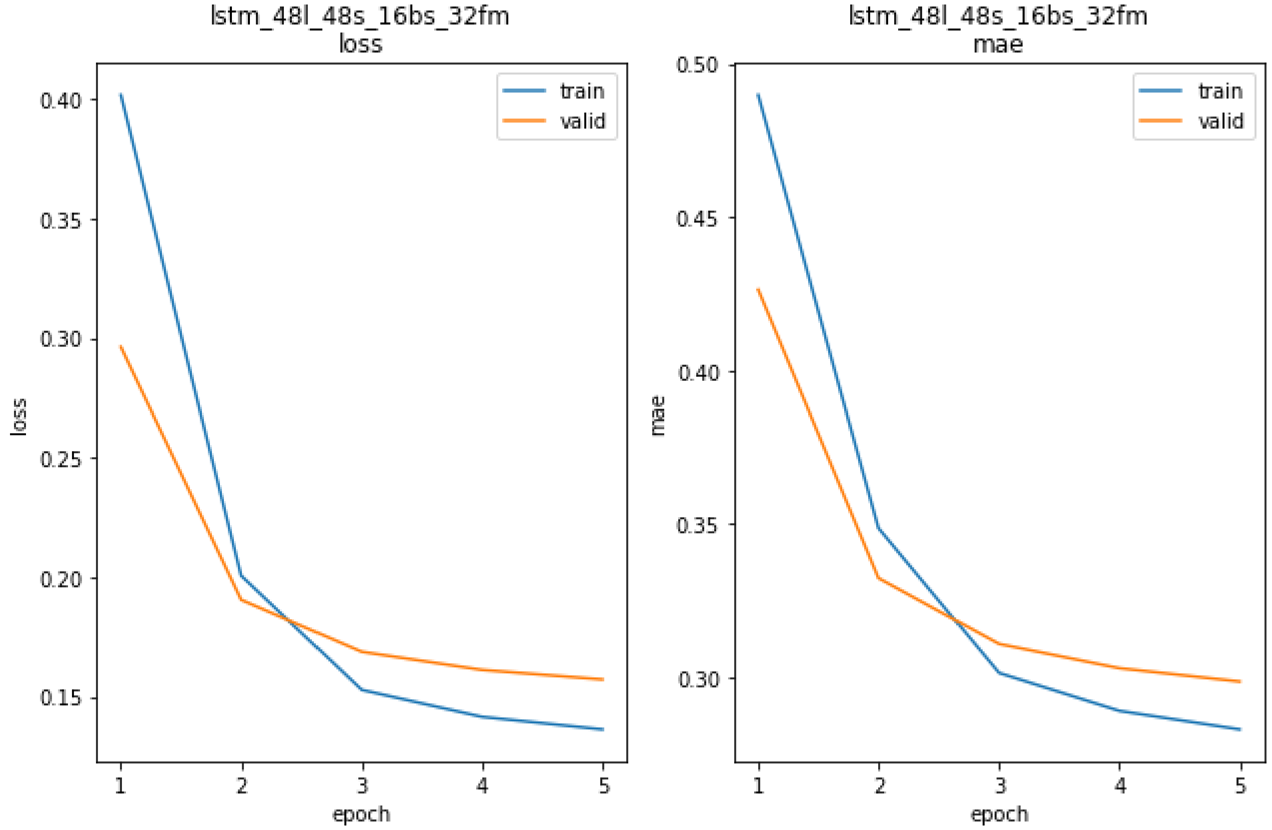



```
lstm_48l_48s_16bs_32fm train min loss: 0.130340 mae: 0.275672   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.147361 mae: 0.290022   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'level'
Epoch 1/5
11758/11758 [==============================] - 14s 1ms/step - loss: 2.1348 - m
```

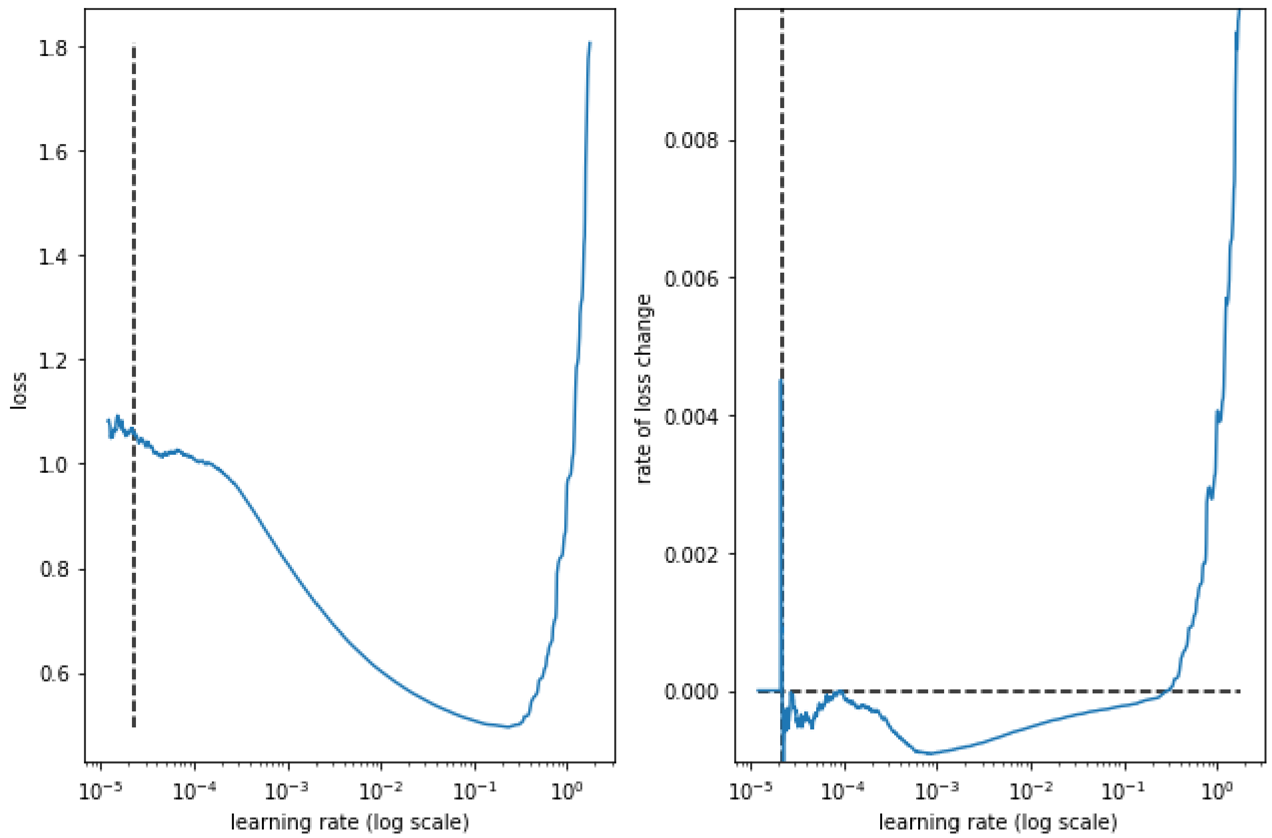



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_49 (LSTM)              (None, 32)                5248

 dense_586 (Dense)           (None, 48)                1584

 reshape_110 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,832
Trainable params: 6,832
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 59s - loss: 0.4016 - mae: 0.4898 - val_loss: 0.2962 - val_mae: 0
Epoch 2/5
11758/11758 - 58s - loss: 0.2007 - mae: 0.3487 - val_loss: 0.1906 - val_mae: 0
```

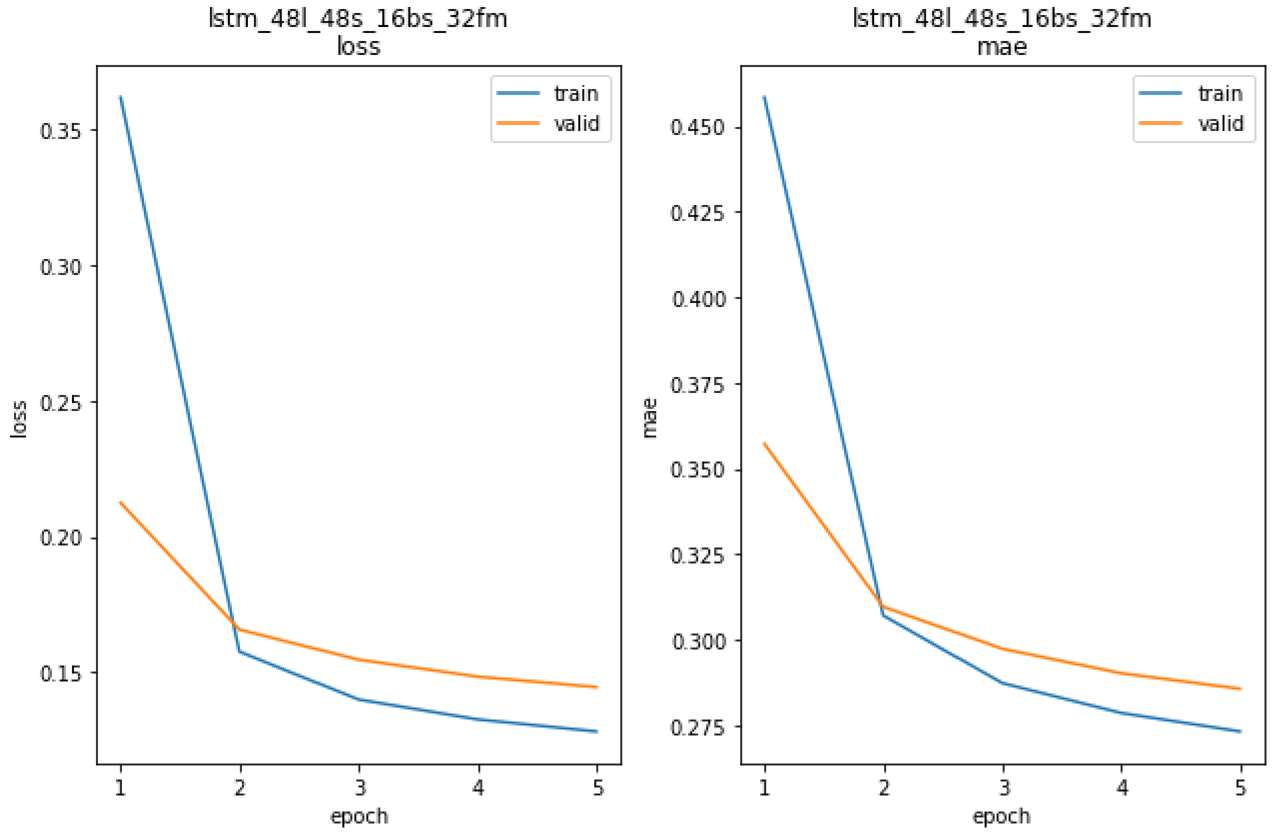
```
Epoch 3/5
11758/11758 - 57s - loss: 0.1529 - mae: 0.3015 - val_loss: 0.1688 - val_mae: 0
Epoch 4/5
11758/11758 - 58s - loss: 0.1416 - mae: 0.2891 - val_loss: 0.1612 - val_mae: 0
Epoch 5/5
11758/11758 - 57s - loss: 0.1364 - mae: 0.2831 - val_loss: 0.1573 - val_mae: 0
```




```
lstm_48l_48s_16bs_32fm train min loss: 0.136358 mae: 0.283101   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.157250 mae: 0.298707   epoch: 5

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'seasor
Epoch 1/5
11758/11758 [==============================] - 14s 1ms/step - loss: 1.9893 - m
```



```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_50 (LSTM)              (None, 32)                5248

 dense_587 (Dense)           (None, 48)                1584

 reshape_111 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,832
Trainable params: 6,832
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
11758/11758 - 58s - loss: 0.3620 - mae: 0.4584 - val_loss: 0.2125 - val_mae: 0
Epoch 2/5
11758/11758 - 57s - loss: 0.1576 - mae: 0.3072 - val_loss: 0.1657 - val_mae: 0
Epoch 3/5
11758/11758 - 57s - loss: 0.1399 - mae: 0.2874 - val_loss: 0.1546 - val_mae: 0
Epoch 4/5
11758/11758 - 57s - loss: 0.1325 - mae: 0.2787 - val_loss: 0.1483 - val_mae: 0
Epoch 5/5
11758/11758 - 57s - loss: 0.1281 - mae: 0.2733 - val_loss: 0.1445 - val_mae: 0
```




```
lstm_48l_48s_16bs_32fm train min loss: 0.128083 mae: 0.273261   epoch: 5
lstm_48l_48s_16bs_32fm valid min loss: 0.144460 mae: 0.285737   epoch: 5

[('lstm_48l_48s_16bs_32fm', 0.14446),
 ('lstm_48l_48s_16bs_32fm_tbats', 0.14486)]
[('lstm_48l_48s_16bs_32fm', 0.28574),
 ('lstm_48l_48s_16bs_32fm_tbats', 0.28699)]
CPU times: user 1h 15min 43s, sys: 8min 17s, total: 1h 24min 1s
Wall time: 1h 5min 4s
```

```
Wall time: 1h 5min 1s
```

Results for feature selection runs (48 steps ahead, 5 epochs):

xcols	features	mse	mae
def_cols	y, humidity, dew.point, pressure, wind.x, wind.y, day.sin, day.cos, year.sin, year.cos	0.13953	0.27967
y_col	y	0.19447	0.33524
notime	y, humidity, dew.point, pressure, wind.x, wind.y	0.16726	0.30773
nowind	y, humidity, dew.point, pressure, day.sin, day.cos, year.sin, year.cos	0.14898	0.28986
var_cols	y, humidity, dew.point, pressure	0.17672	0.31481
day_col	y, humidity, dew.point, pressure, wind.x, wind.y, day.sin, day.cos	0.16022	0.29916
year_col	y, humidity, dew.point, pressure, wind.x, wind.y, year.sin, year.cos	0.17612	0.31477
tbats_cols	y, humidity, dew.point, pressure, wind.x, wind.y, level, season1, season2	0.14487	0.28699
tbats_day	y, humidity, dew.point, pressure, wind.x, wind.y, level, season1	0.14736	0.29002
tbats_year	y, humidity, dew.point, pressure, wind.x, wind.y, level, season2	0.15725	0.29871
tbats_nolevel	y, humidity, dew.point, pressure, wind.x, wind.y, season1, season2	0.14446	0.28574

Using def_cols gives the minimal mean squared error values of 0.1395.

The tbats_cols and tbats_nolevel also gave low mse values of 0.1449 and 0.1445 respectively.

The two tbats_nolevel time components (season1 and season2) perform quite well compared to the four default time components.

In unrelated news, it's good to see there is probably some signal in the wind vectors!

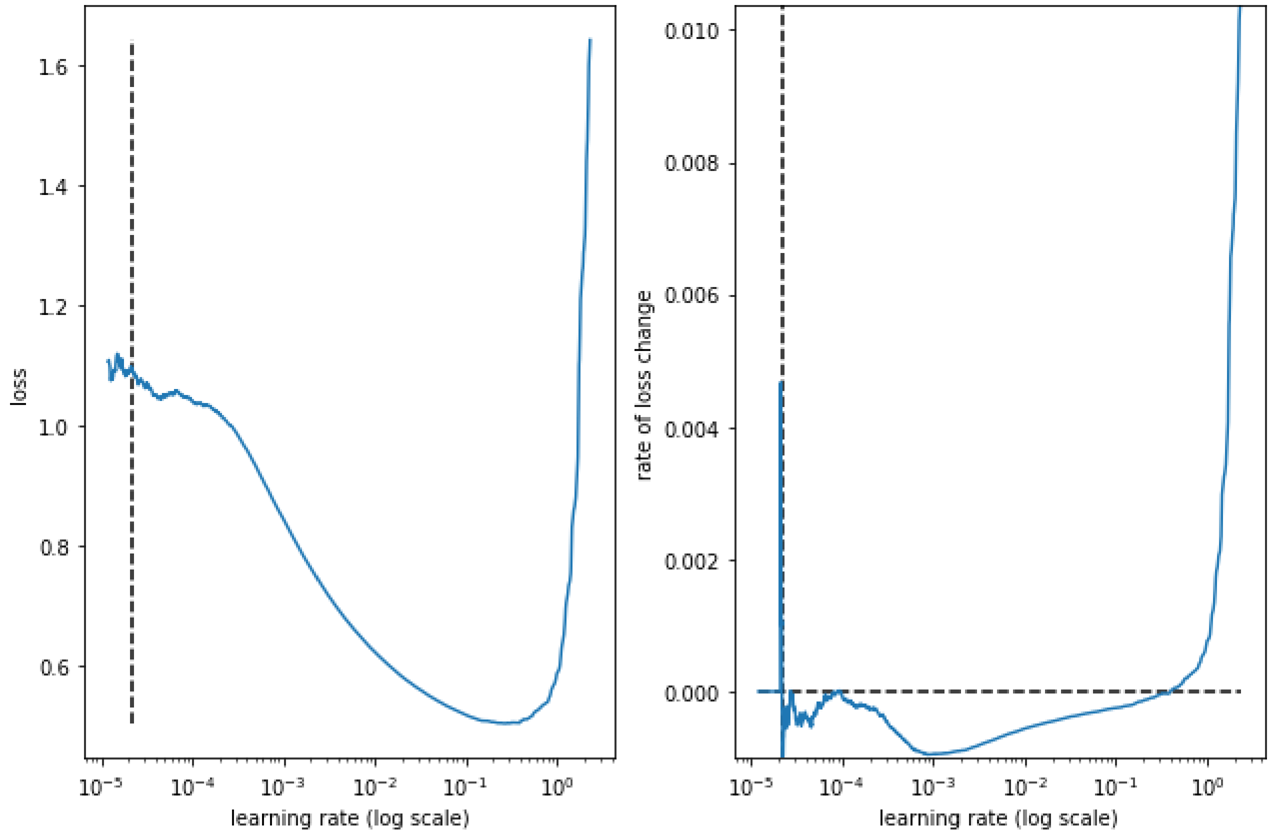
It's clear from some of the learning rate finder curves that start_lr and/or end_lr could be further refined. Start_lr seems low.

Here I compare the def_cols and tbats_nolevel time components over 20 epochs.

```
%%time


# def_cols      = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y',
# tbats_nolevel = ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y',

params = get_default_params('lstm')
params.update({'epochs': 20})

sweep_values = {'xcols': [def_cols, tbats_nolevel]}
models, xcol_model_names = sweep_param(models, params, sweep_values, verbose=True)
```

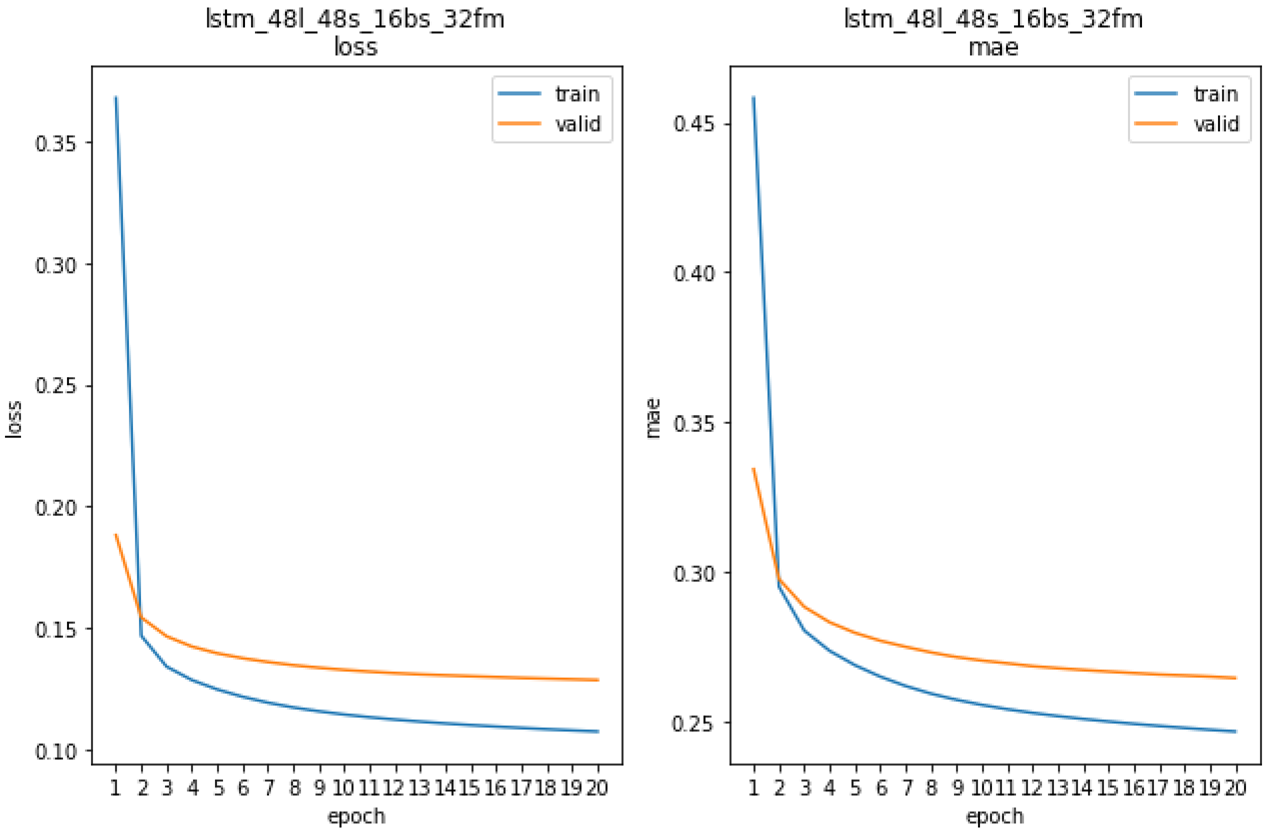
```
get_best_models(models, xcol_model_names)
get_best_models(models)

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

```
xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'day.si
Epoch 1/5
11758/11758 [==============================] - 13s 949us/step - loss: 2.0547 -
```

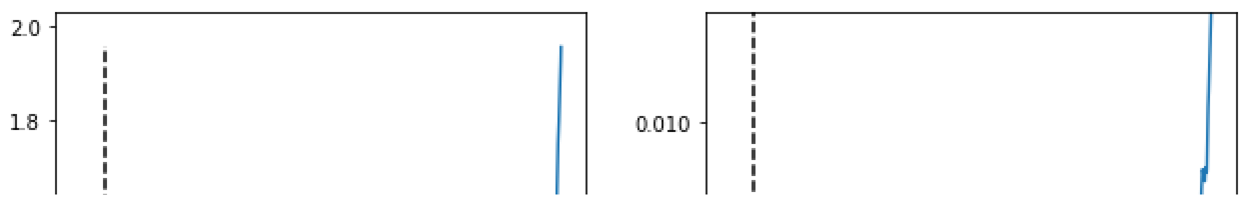


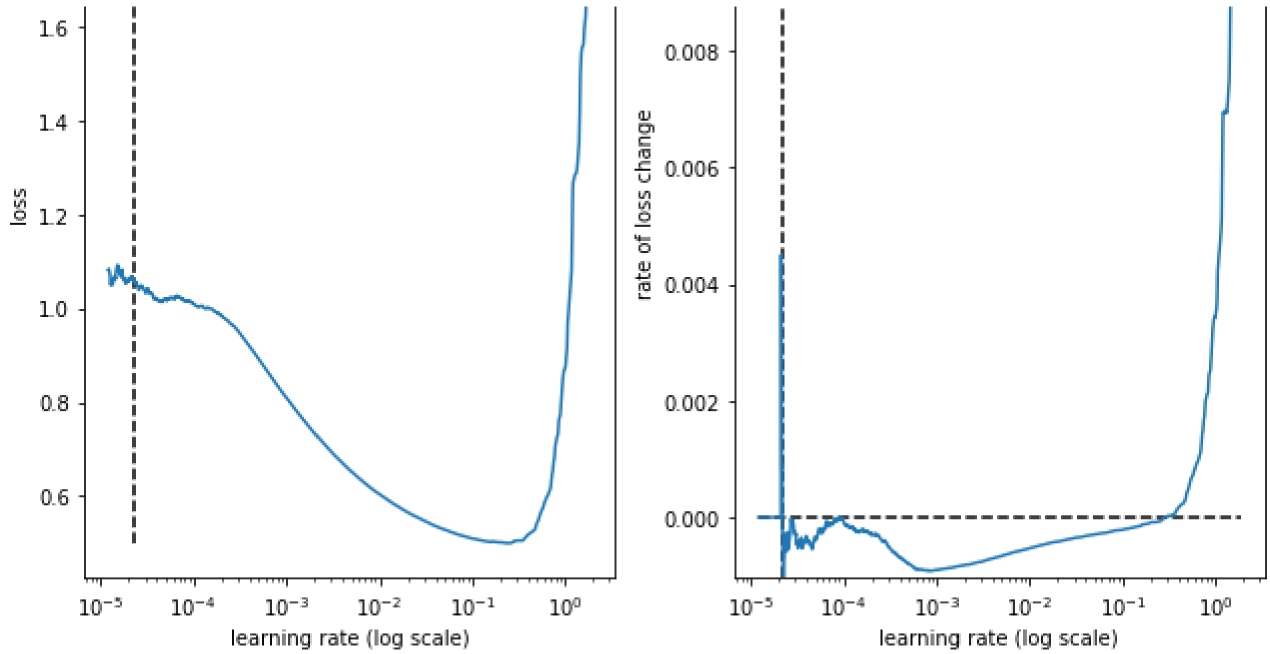

```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_51 (LSTM)              (None, 32)                5504

 dense_588 (Dense)           (None, 48)                1584

 reshape_112 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
11758/11758 - 61s - loss: 0.3681 - mae: 0.4582 - val_loss: 0.1882 - val_mae: 0
Epoch 2/20
11758/11758 - 58s - loss: 0.1468 - mae: 0.2951 - val_loss: 0.1543 - val_mae: 0
Epoch 3/20
11758/11758 - 58s - loss: 0.1341 - mae: 0.2804 - val_loss: 0.1465 - val_mae: 0
Epoch 4/20
11758/11758 - 60s - loss: 0.1285 - mae: 0.2736 - val_loss: 0.1423 - val_mae: 0
Epoch 5/20
11758/11758 - 60s - loss: 0.1247 - mae: 0.2689 - val_loss: 0.1395 - val_mae: 0
Epoch 6/20
11758/11758 - 58s - loss: 0.1216 - mae: 0.2650 - val_loss: 0.1375 - val_mae: 0
Epoch 7/20
11758/11758 - 58s - loss: 0.1192 - mae: 0.2619 - val_loss: 0.1359 - val_mae: 0
Epoch 8/20
11758/11758 - 60s - loss: 0.1172 - mae: 0.2594 - val_loss: 0.1346 - val_mae: 0
```

```
11758/11758 - 60s - loss: 0.1172 - mae: 0.2594 - val_loss: 0.1346 - val_mae: 0
Epoch 9/20
11758/11758 - 60s - loss: 0.1156 - mae: 0.2573 - val_loss: 0.1336 - val_mae: 0
Epoch 10/20
11758/11758 - 58s - loss: 0.1143 - mae: 0.2556 - val_loss: 0.1327 - val_mae: 0
Epoch 11/20
11758/11758 - 58s - loss: 0.1132 - mae: 0.2541 - val_loss: 0.1320 - val_mae: 0
Epoch 12/20
11758/11758 - 58s - loss: 0.1122 - mae: 0.2529 - val_loss: 0.1314 - val_mae: 0
Epoch 13/20
11758/11758 - 58s - loss: 0.1114 - mae: 0.2518 - val_loss: 0.1309 - val_mae: 0
Epoch 14/20
11758/11758 - 58s - loss: 0.1106 - mae: 0.2509 - val_loss: 0.1305 - val_mae: 0
Epoch 15/20
11758/11758 - 58s - loss: 0.1100 - mae: 0.2500 - val_loss: 0.1301 - val_mae: 0
Epoch 16/20
11758/11758 - 58s - loss: 0.1093 - mae: 0.2493 - val_loss: 0.1297 - val_mae: 0
Epoch 17/20
11758/11758 - 58s - loss: 0.1088 - mae: 0.2485 - val_loss: 0.1294 - val_mae: 0
Epoch 18/20
11758/11758 - 58s - loss: 0.1083 - mae: 0.2479 - val_loss: 0.1291 - val_mae: 0
Epoch 19/20
11758/11758 - 58s - loss: 0.1078 - mae: 0.2473 - val_loss: 0.1288 - val_mae: 0
Epoch 20/20
11758/11758 - 58s - loss: 0.1073 - mae: 0.2467 - val_loss: 0.1285 - val_mae: 0
```



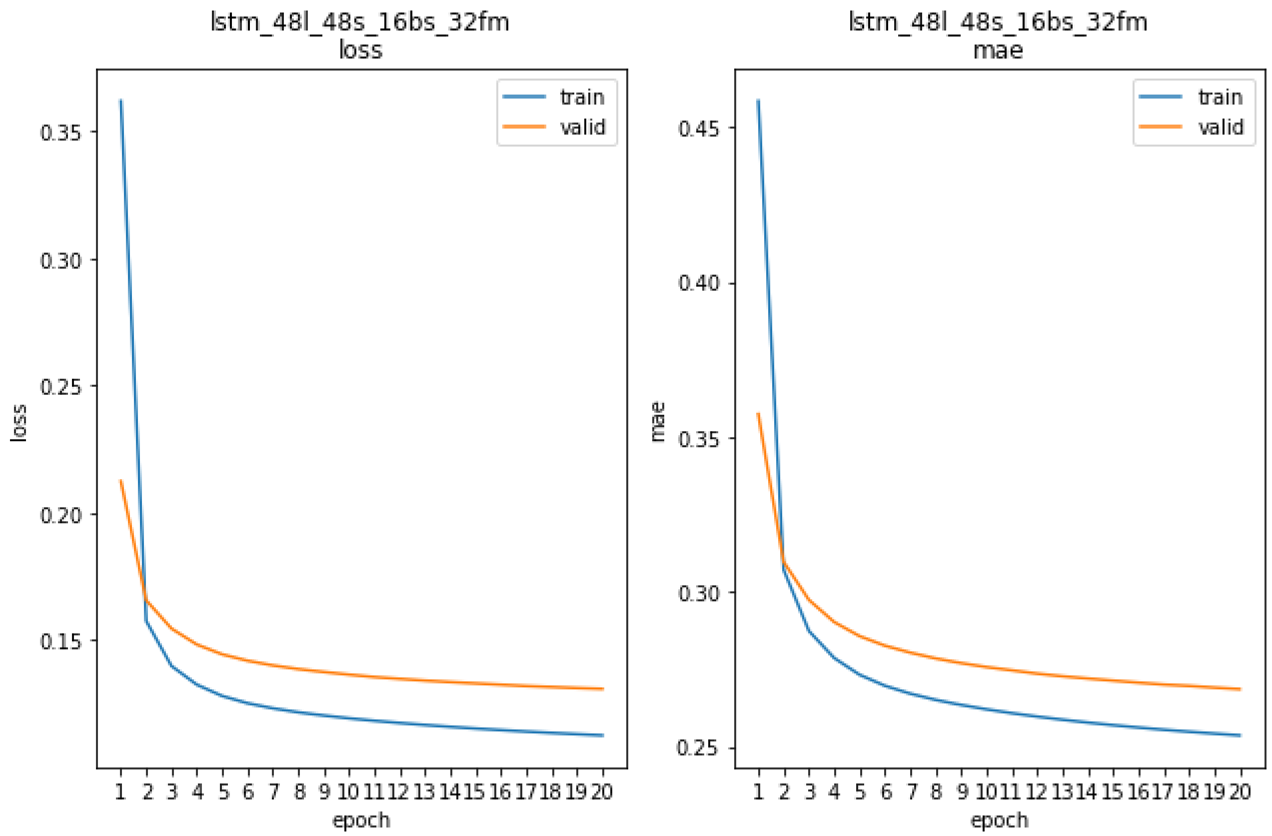

```
lstm_48l_48s_16bs_32fm train min loss: 0.107322 mae: 0.246702   epoch: 20
lstm_48l_48s_16bs_32fm valid min loss: 0.128516 mae: 0.264562   epoch: 20

xcols : ['y', 'humidity', 'dew.point', 'pressure', 'wind.x', 'wind.y', 'season
Epoch 1/5
11758/11758 [==============================] - 13s 938us/step - loss: 2.0185 -
```






```
best lr: 2.2358741e-05

Model: "lstm_48l_48s_16bs_32fm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_52 (LSTM)              (None, 32)                5248

 dense_589 (Dense)           (None, 48)                1584

 reshape_113 (Reshape)       (None, 48, 1)             0

=================================================================
Total params: 6,832
Trainable params: 6,832
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
11758/11758 - 60s - loss: 0.3620 - mae: 0.4584 - val_loss: 0.2125 - val_mae: 0
Epoch 2/20
11758/11758 - 58s - loss: 0.1576 - mae: 0.3072 - val_loss: 0.1657 - val_mae: 0
Epoch 3/20
11758/11758 - 57s - loss: 0.1399 - mae: 0.2874 - val_loss: 0.1546 - val_mae: 0
Epoch 4/20
11758/11758 - 58s - loss: 0.1325 - mae: 0.2787 - val_loss: 0.1483 - val_mae: 0
Epoch 5/20
11758/11758 - 59s - loss: 0.1281 - mae: 0.2733 - val_loss: 0.1445 - val_mae: 0
Epoch 6/20
11758/11758 - 58s - loss: 0.1252 - mae: 0.2697 - val_loss: 0.1419 - val_mae: 0
Epoch 7/20
11758/11758 - 58s - loss: 0.1232 - mae: 0.2672 - val_loss: 0.1401 - val_mae: 0
Epoch 8/20
11758/11758 - 60s - loss: 0.1217 - mae: 0.2652 - val_loss: 0.1387 - val_mae: 0
Epoch 9/20
11758/11758 - 58s - loss: 0.1204 - mae: 0.2635 - val_loss: 0.1375 - val_mae: 0
Epoch 10/20
11758/11758 - 59s - loss: 0.1193 - mae: 0.2621 - val_loss: 0.1365 - val_mae: 0
Epoch 11/20
11758/11758 - 57s - loss: 0.1183 - mae: 0.2608 - val_loss: 0.1355 - val_mae: 0
Epoch 12/20
11758/11758 - 58s - loss: 0.1174 - mae: 0.2597 - val_loss: 0.1348 - val_mae: 0
Epoch 13/20
```

```
Epoch 13/20
11758/11758 - 58s - loss: 0.1167 - mae: 0.2587 - val_loss: 0.1341 - val_mae: 0
Epoch 14/20
11758/11758 - 58s - loss: 0.1159 - mae: 0.2578 - val_loss: 0.1335 - val_mae: 0
Epoch 15/20
11758/11758 - 58s - loss: 0.1153 - mae: 0.2570 - val_loss: 0.1330 - val_mae: 0
Epoch 16/20
11758/11758 - 58s - loss: 0.1147 - mae: 0.2562 - val_loss: 0.1325 - val_mae: 0
Epoch 17/20
11758/11758 - 59s - loss: 0.1141 - mae: 0.2555 - val_loss: 0.1320 - val_mae: 0
Epoch 18/20
11758/11758 - 58s - loss: 0.1135 - mae: 0.2549 - val_loss: 0.1316 - val_mae: 0
Epoch 19/20
11758/11758 - 58s - loss: 0.1130 - mae: 0.2543 - val_loss: 0.1312 - val_mae: 0
Epoch 20/20
11758/11758 - 58s - loss: 0.1126 - mae: 0.2537 - val_loss: 0.1309 - val_mae: 0
```




```
lstm_48l_48s_16bs_32fm train min loss: 0.112574 mae: 0.253682   epoch: 20
lstm_48l_48s_16bs_32fm valid min loss: 0.130871 mae: 0.268615   epoch: 20

[('lstm_48l_48s_16bs_32fm', 0.13087),
 ('lstm_48l_48s_16bs_32fm_tbats', 0.14486)]
[('lstm_48l_48s_16bs_32fm', 0.26861),
 ('lstm_48l_48s_16bs_32fm_tbats', 0.28699)]
CPU times: user 54min 24s, sys: 5min 57s, total: 1h 22s
Wall time: 47min 15s
```

Results for def_cols and tbats_nolevel time components over 20 epochs:

xcols	features	mse	mae
def_cols	y, humidity, dew.point, pressure, wind.x, wind.y, day.sin, day.cos, year.sin, year.cos	0.12852	0.26456
tbats_nolevel	y, humidity, dew.point, pressure, wind.x, wind.y, season1, season2	0.13087	0.26861

As before, the sinusoidal time components give lower mse and mae values.

It may be possible to further reduce the TBATS mse value by correcting the start/end of year boundary mis-match problem. Given more time the next option worth trying is the Short Time Fourier Transform probably from scipy and/or wavelets. For now, I'll continue with the default time components (daily/yearly sin/cos).

Start_lr seems low.

With mixup

I tested a range of mixup alpha values (1,2,3,4,5) but they had negligible affect on the mse values. For brevity, I've removed these tests. The mixup paper recommends an alpha value of 4, which I use throughout this notebook.

Next, I use Bayesian optimisation from the scikit-optimize package to select optimal values from:

- mix_type - time-series
- mix_factor - how much mixup augmentation, 1 or 2
- mix_diff - time difference for time series mixup, 1 to 48

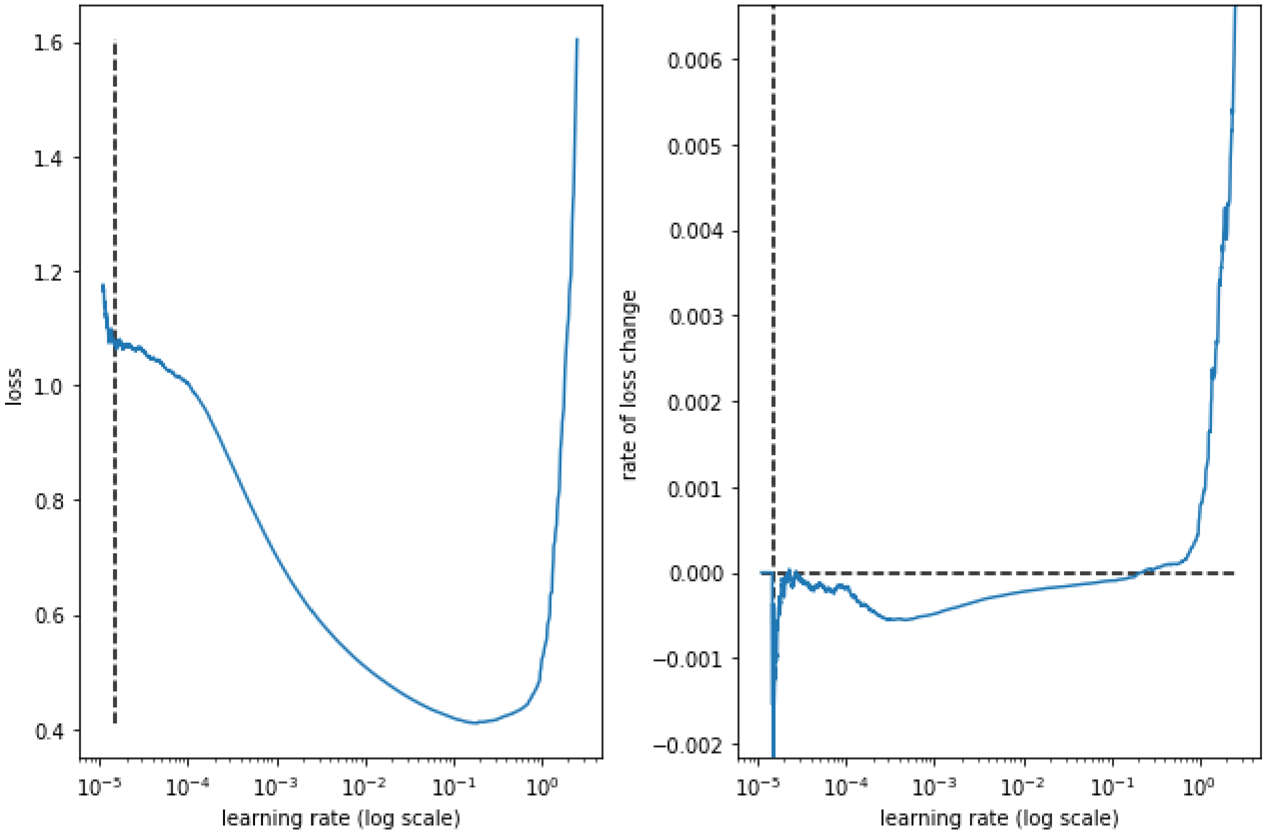
Note:

- I limit mix_factor to 2 to minimise compute time
- mix_diff does not apply to input mixup
- 'input' mixup will follow in a subsequent cell for comparison

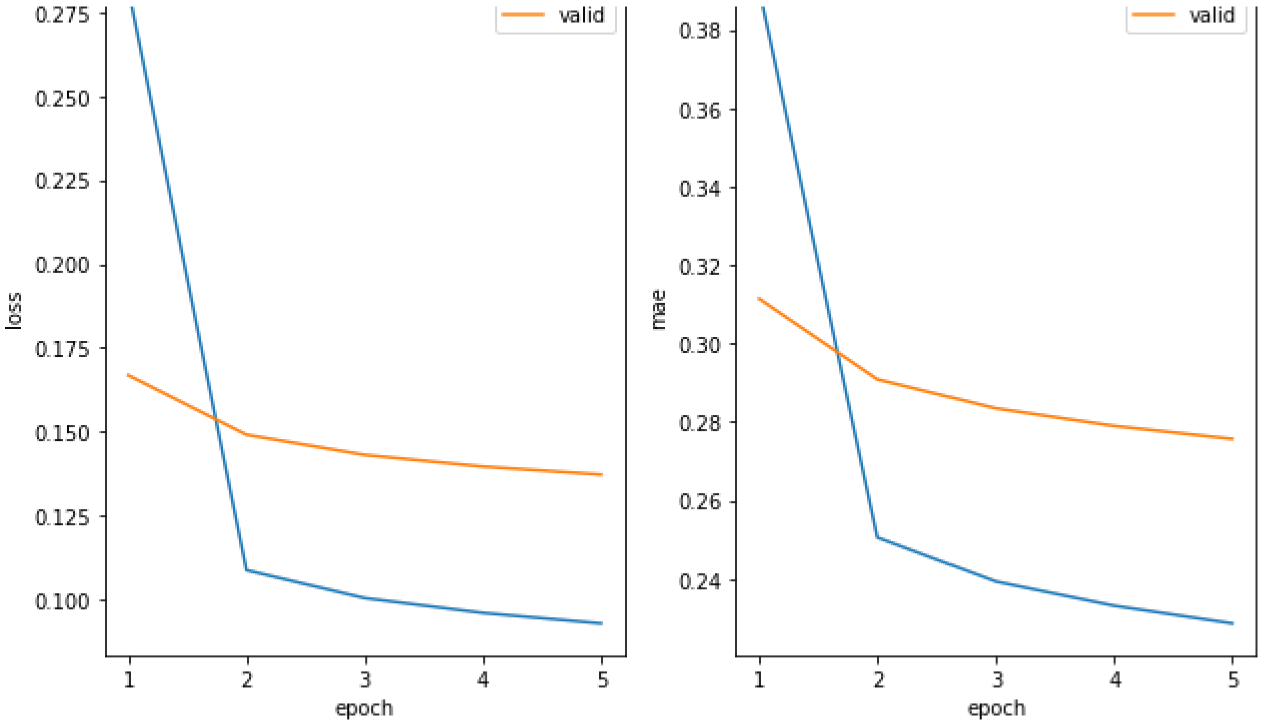
```
%%time


dim_mix_factor = Integer(low = 1, high =  2, name = 'mix_factor')
dim_mix_diff   = Integer(low = 1, high = 48, name = 'mix_diff')

bo_dims_lstm_48s_mixup = [dim_mix_factor,
                          dim_mix_diff]


@use_named_args(dimensions = bo_dims_lstm_48s_mixup)
def model_fitness_lstm_48s_mixup(**dims):
    params = get_default_params('lstm')

    return get_bo_mse(params, **dims)


bo_def_dims_lstm_48s_mixup = [1, 48]
bo_results_id = 'lstm_48s_mixup'

bo_search_results[bo_results_id] = run_bo_search(model_fitness_lstm_48s_mixup,
                                                 bo_dims_lstm_48s_mixup,
```

```
bo_def_dims_lstm_48s_mixup,
20)
```

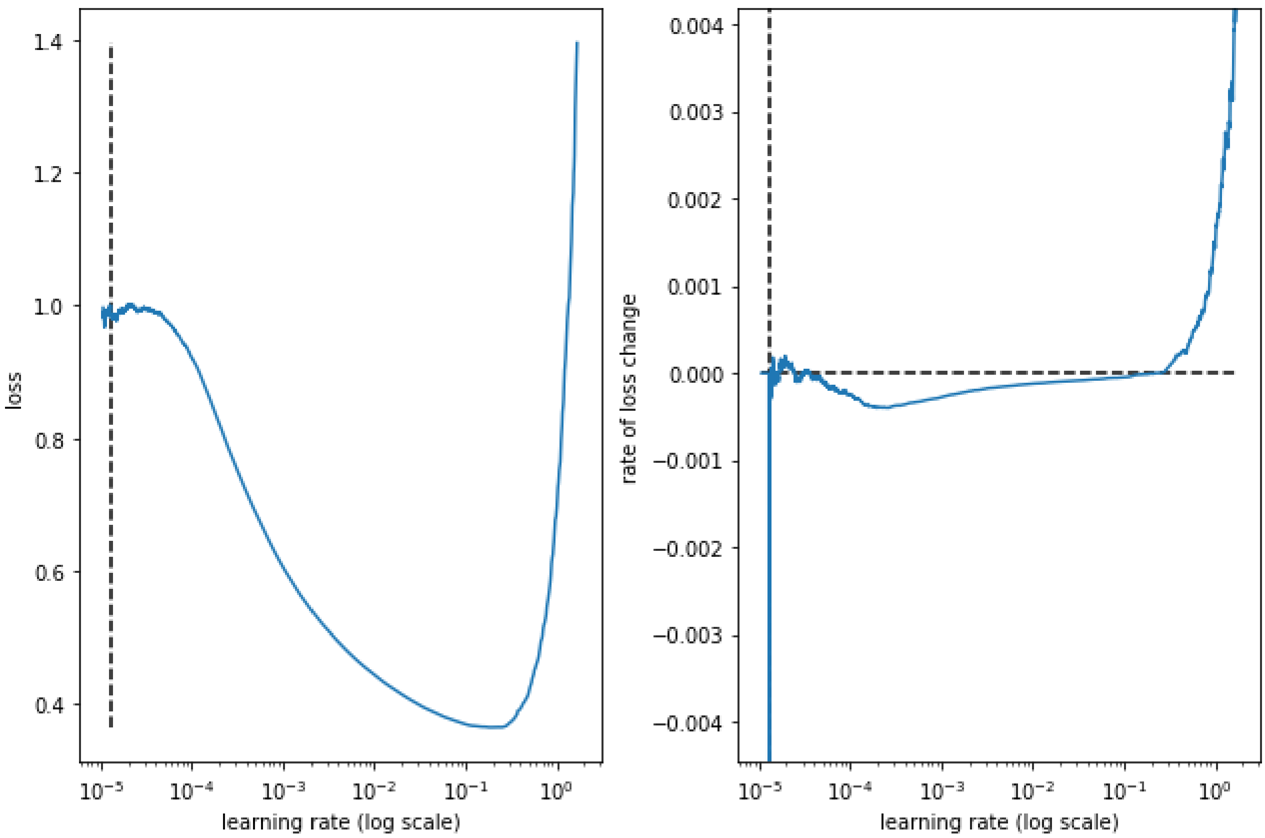
```
Iteration No: 1 started. Evaluating function at provided point.
mix_factor 1
mix_diff 48
Epoch 1/5
23518/23518 [==============================] - 25s 873us/step - loss: 1.6634 -
```




```
best lr: 1.4952328e-05

Model: "lstm_48l_48s_16bs_32fm_1m_4a_48td"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 32)                5504

 dense (Dense)               (None, 48)                1584

 reshape (Reshape)           (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
23518/23518 - 107s - loss: 0.2820 - mae: 0.3904 - val_loss: 0.1668 - val_mae:
Epoch 2/5
23518/23518 - 109s - loss: 0.1087 - mae: 0.2506 - val_loss: 0.1491 - val_mae:
Epoch 3/5
23518/23518 - 107s - loss: 0.1004 - mae: 0.2394 - val_loss: 0.1431 - val_mae:
Epoch 4/5
23518/23518 - 106s - loss: 0.0959 - mae: 0.2332 - val_loss: 0.1396 - val_mae:
Epoch 5/5
23518/23518 - 109s - loss: 0.0928 - mae: 0.2287 - val_loss: 0.1372 - val_mae:
```

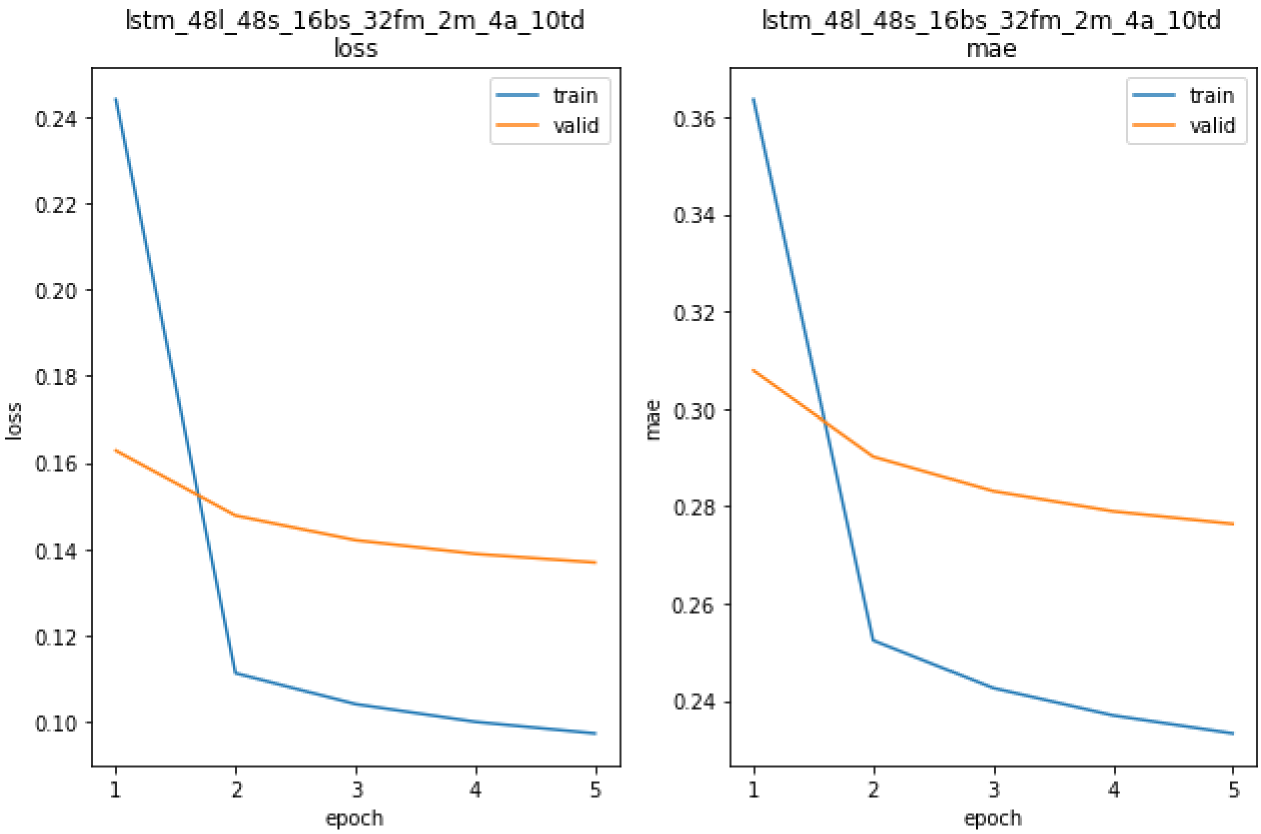



```
lstm_48l_48s_16bs_32fm_1m_4a_48td train min loss: 0.092809      mae: 0.228722
lstm_48l_48s_16bs_32fm_1m_4a_48td valid min loss: 0.137203      mae: 0.275726

lstm_48l_48s_16bs_32fm_1m_4a_48td
Iteration No: 1 ended. Evaluation done at provided point.
Time taken: 599.5275
Function value obtained: 0.1372
Current minimum: 0.1372
Iteration No: 2 started. Evaluating function at random point.
mix_factor 2
mix_diff 10
Epoch 1/5
35283/35283 [==============================] - 33s 878us/step - loss: 1.4648 -
```



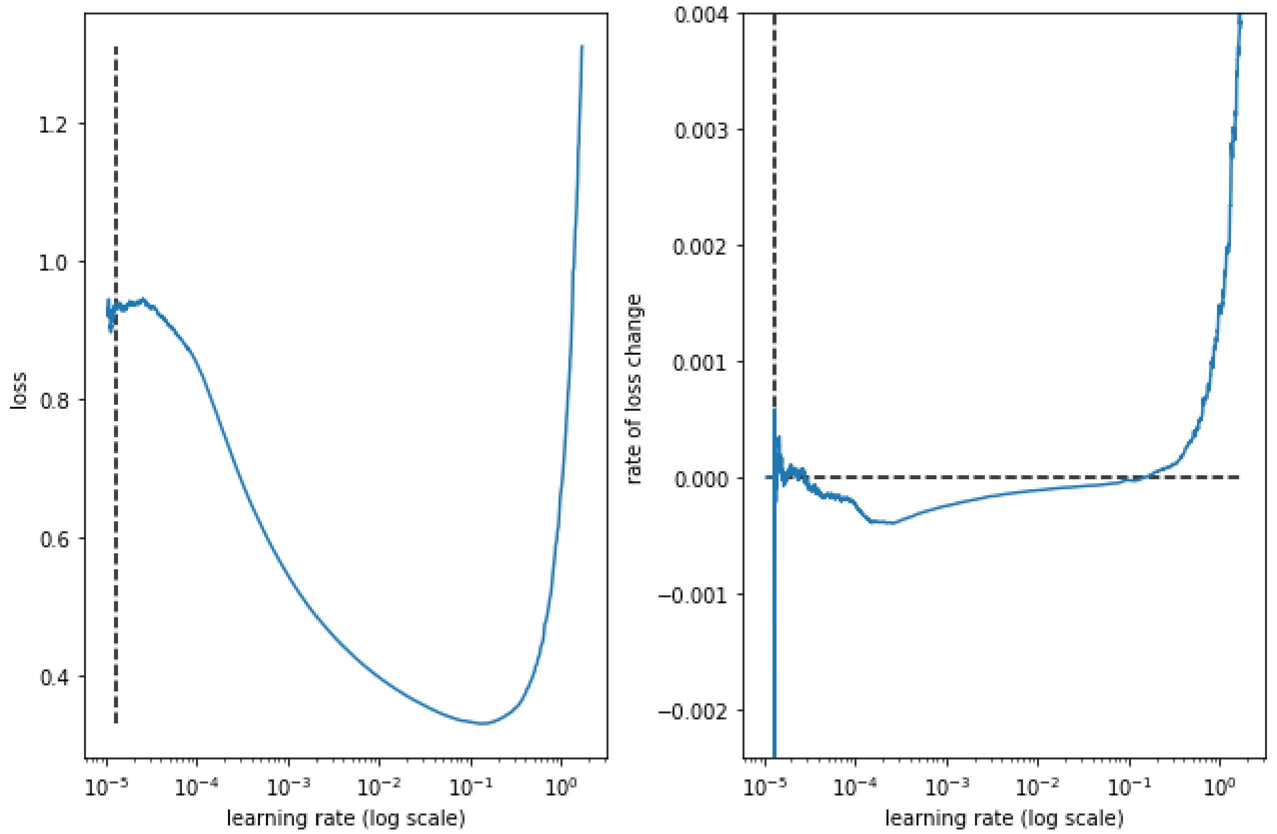

```
best lr: 1.2847962e-05
```

```
Model: "lstm_48l_48s_16bs_32fm_2m_4a_10td"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 32)                5504

 dense_1 (Dense)             (None, 48)                1584

 reshape_1 (Reshape)         (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
35283/35283 - 159s - loss: 0.2441 - mae: 0.3636 - val_loss: 0.1628 - val_mae:
Epoch 2/5
35283/35283 - 156s - loss: 0.1112 - mae: 0.2524 - val_loss: 0.1478 - val_mae:
Epoch 3/5
35283/35283 - 156s - loss: 0.1040 - mae: 0.2426 - val_loss: 0.1420 - val_mae:
Epoch 4/5
35283/35283 - 155s - loss: 0.0999 - mae: 0.2370 - val_loss: 0.1389 - val_mae:
Epoch 5/5
35283/35283 - 156s - loss: 0.0972 - mae: 0.2333 - val_loss: 0.1369 - val_mae:
```

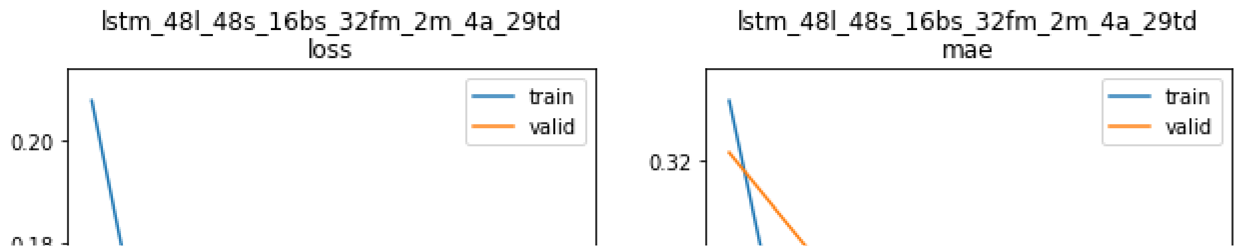


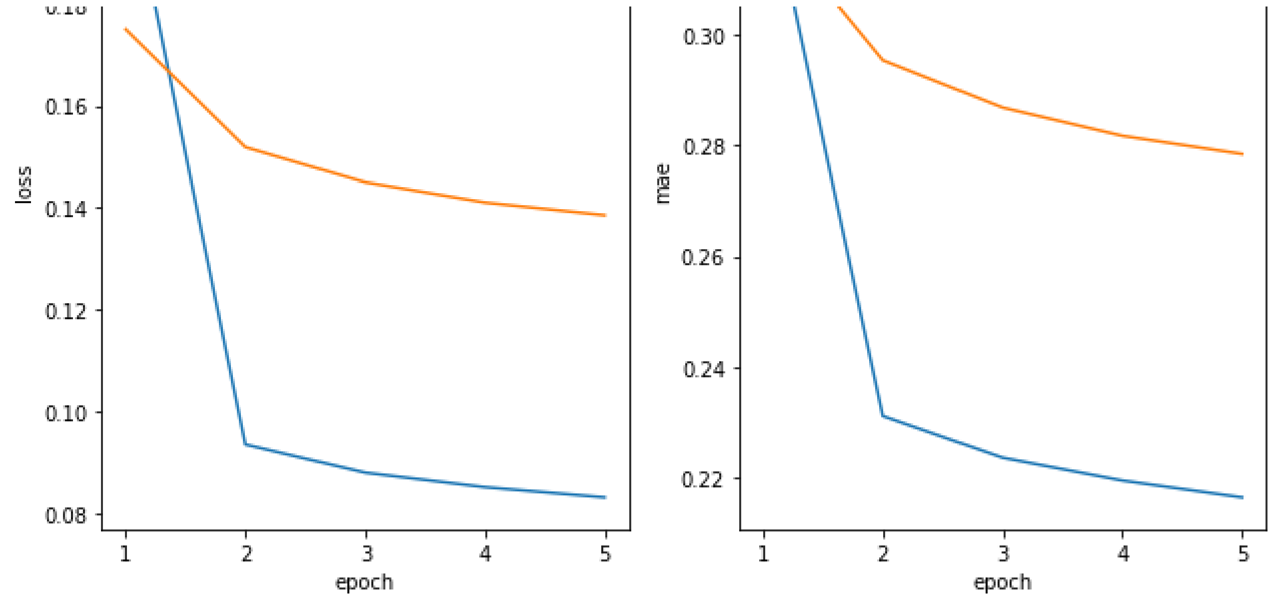

```
lstm_48l_48s_16bs_32fm_2m_4a_10td train min loss: 0.097233      mae: 0.233282
lstm_48l_48s_16bs_32fm_2m_4a_10td valid min loss: 0.136871      mae: 0.276383

lstm_48l_48s_16bs_32fm_2m_4a_10td
Iteration No: 2 ended. Evaluation done at random point.
Time taken: 964.3083
Function value obtained: 0.1369
Current minimum: 0.1369
Iteration No: 3 started. Evaluating function at random point.
mix_factor 2
mix_diff 29
```

```
Epoch 1/5
35280/35280 [==============================] - 34s 910us/step - loss: 1.3229 -
```

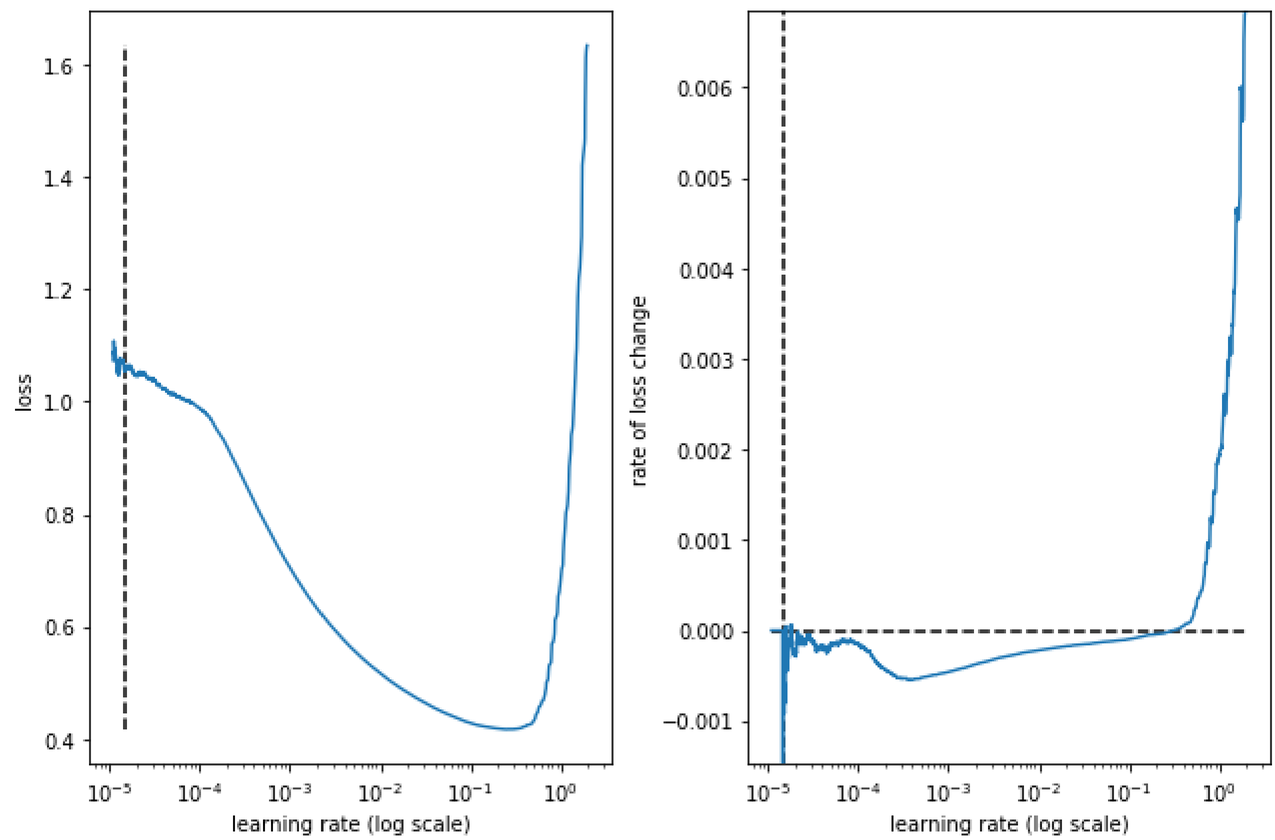



```
best lr: 1.2880487e-05

Model: "lstm_48l_48s_16bs_32fm_2m_4a_29td"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 32)                5504

 dense_2 (Dense)             (None, 48)                1584

 reshape_2 (Reshape)         (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
35280/35280 - 162s - loss: 0.2078 - mae: 0.3309 - val_loss: 0.1750 - val_mae:
Epoch 2/5
35280/35280 - 159s - loss: 0.0935 - mae: 0.2312 - val_loss: 0.1519 - val_mae:
Epoch 3/5
35280/35280 - 160s - loss: 0.0879 - mae: 0.2237 - val_loss: 0.1450 - val_mae:
Epoch 4/5
35280/35280 - 159s - loss: 0.0851 - mae: 0.2196 - val_loss: 0.1410 - val_mae:
Epoch 5/5
35280/35280 - 160s - loss: 0.0831 - mae: 0.2165 - val_loss: 0.1386 - val_mae:
```



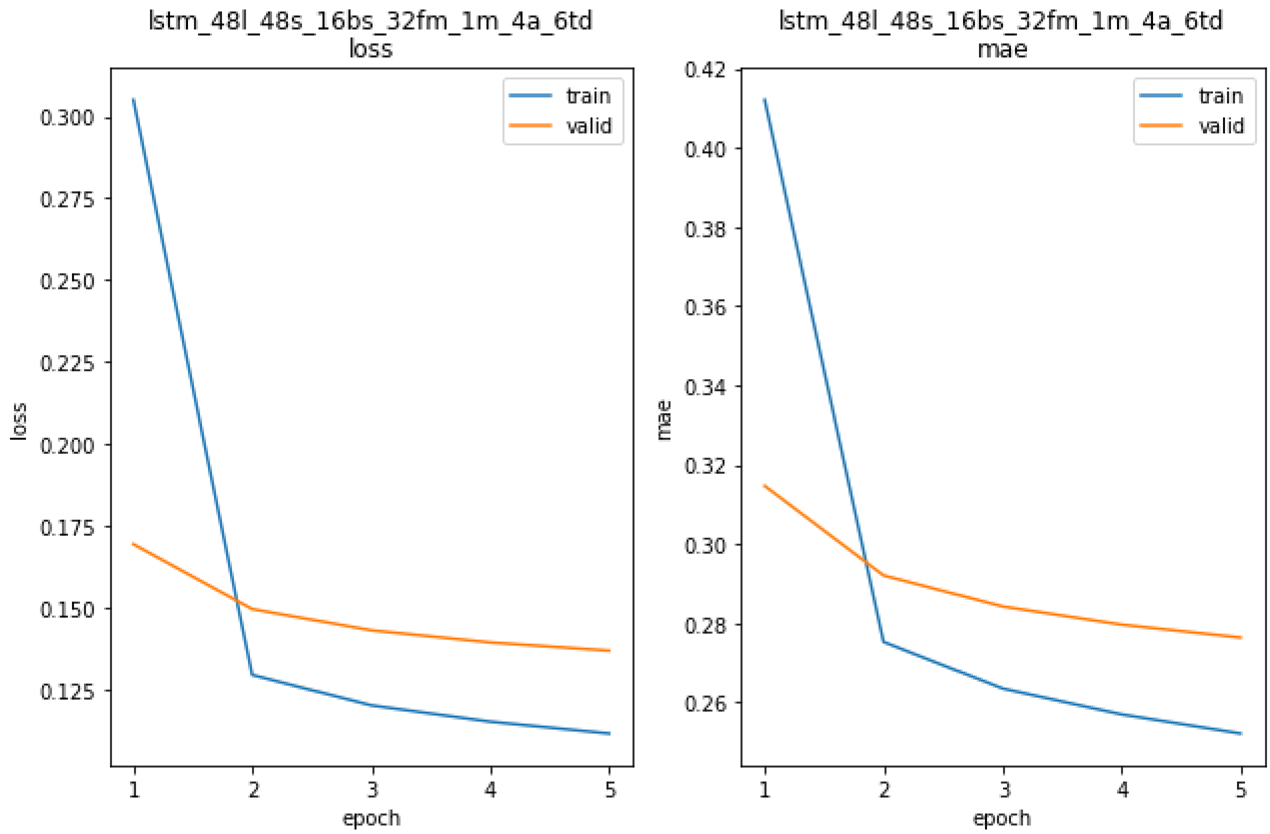



```
lstm_48l_48s_16bs_32fm_2m_4a_29td train min loss: 0.083089      mae: 0.216545
lstm_48l_48s_16bs_32fm_2m_4a_29td valid min loss: 0.138555      mae: 0.278403

lstm_48l_48s_16bs_32fm_2m_4a_29td
Iteration No: 3 ended. Evaluation done at random point.
Time taken: 962.8563
Function value obtained: 0.1386
Current minimum: 0.1369
Iteration No: 4 started. Evaluating function at random point.
mix_factor 1
mix_diff 6
Epoch 1/5
23520/23520 [==============================] - 24s 942us/step - loss: 1.6727 -
```




```
best lr: 1.4618337e-05

Model: "lstm_48l_48s_16bs_32fm_1m_4a_6td"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 lstm_3 (LSTM)              (None, 32)                5504

 dense_3 (Dense)            (None, 48)                1584

 reshape_3 (Reshape)        (None, 48, 1)             0

=================================================================
Total params: 7,088
Trainable params: 7,088
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
23520/23520 - 110s - loss: 0.3051 - mae: 0.4121 - val_loss: 0.1693 - val_mae:
Epoch 2/5
23520/23520 - 108s - loss: 0.1293 - mae: 0.2753 - val_loss: 0.1495 - val_mae:
Epoch 3/5
23520/23520 - 110s - loss: 0.1200 - mae: 0.2635 - val_loss: 0.1429 - val_mae:
Epoch 4/5
23520/23520 - 108s - loss: 0.1150 - mae: 0.2570 - val_loss: 0.1392 - val_mae:
Epoch 5/5
23520/23520 - 107s - loss: 0.1114 - mae: 0.2522 - val_loss: 0.1367 - val_mae:
```




```
lstm_48l_48s_16bs_32fm_1m_4a_6td train min loss: 0.111393       mae: 0.252166
lstm_48l_48s_16bs_32fm_1m_4a_6td valid min loss: 0.136745       mae: 0.276409

lstm_48l_48s_16bs_32fm_1m_4a_6td
Iteration No: 4 ended. Evaluation done at random point.
Time taken: 587.2738
Function value obtained: 0.1367
Current minimum: 0.1367
Iteration No: 5 started. Evaluating function at random point.
mix_factor 1
mix_diff 17
Epoch 1/5
23520/23520 [==============================] - 23s 931us/step - loss: 1.6148 -
```