Machine Learning Engineer Nanodegree

Capstone Project

Report

Ognian Dantchev

August 8th, 2021

Table of Contents

I. Definition

Project Overview

Time series forecasting is applied when scientific predictions are needed based on historical data with time component. Time series forecasting has many applications in various industries. Companies use everything from simple spreadsheets to complex financial planning software in an attempt to accurately forecast future business outcomes such as product demand, resource needs, crops yield, traffic or financial performance. Beyond business, applications include healthcare, medical, environmental studies, social studies and other forecasting.

There are many classical time series forecasting methods, based on autoregressive (AR) models, exponential smoothing, Fourier Transform, etc.

Models based on Neural Networks are able to handle more complex nonlinear patterns. They have less restrictions and make less assumptions; have high predictive power and can be easily automated. Their cons – they require more data; can be more difficult to interpret, and it is more difficult to derive confidence intervals for the forecast.

In this project, three different Neural Networks were implemented for time series forecasting. A Recurrent Neural Network (RNN) with Gated Recurrent Unit (GRU) from the Keras API, as described in the TensorFlow Tutorials in [1]. Second is a Dilated Causal CNN, I implemented in the past in [2] and as originally described in [6] by Aaron van den Oord, Sander Dieleman et al. Both 1st and 2nd models use Keras and TensorFlow 2.5. The third model implemented is based on the DeepAR+ algorithm of Amazon Forecast by AWS.

Problem Statement

The purpose of this project is to implement model based on the DeepAR+ algorithm by Amazon and deploy it to AWS. As a secondary target, I try to compare the results to other NN models for time series. Own dataset with 45 years of local meteorology data was used, but no meteorology domain claims whatsoever.

The implementation was split into 5 smaller Jupyter Notebooks, with file names starting with 01 to 05 respectively:

01. Generate and download the raw dataset at https://www.ncdc.noaa.gov/ . Data import and exploration; Changing some of the labels using the Inventory file. Data cleaning and pre-processing; Feature engineering and data transformation – add day-of-year (between 1 and 366) and the hour-of-day (between 0 and 23); Save the clean data in compact new format for use in next notebooks.

02. Split the data into training, validation, and testing sets; Define and train a Gated Recurrent Unit (GRU) RNN Model.

03. Split the data into training, validation, and testing sets; Define and train a Dilated Causal CNN Model. (model and model2 are the same but use different way of building with Keras functional API – as an experiment).

04. Connect AWS API session, create IAM role for Amazon Forecast, create S3 bucket; upload the dataset to AWS S3 bucket; Define and train a model from AWS Amazon Forecast – Amazon DeepAR+ algorithm. I deviated from the original plan to use the open-source Gluon Time Series (GluonTS) toolkit by AWS Labs, in favor of the Amazon DeepAR+ algorithm, as it is state-of-the-art and used by amazon.com; Deployed the later model to AWS.

05. Discussion, TODO, Cleanup of AWS resources and References. Evaluate and compare the models, to the extent possible.

Metrics

Common metrics for Time Series Analysis are Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), Weighted Average Percentage Error (WAPE, also referred to as the MAD/Mean ratio).

$$MSE = \frac{1}{n}\sum_{j=0}^{n}(y_i - \widehat{y_i})$$

Mean Squared Error (MSE) will be used as the loss-function that will be minimized and used as a metric. This measures how closely the model's output matches the true output signals.

At the beginning of a sequence, the model has only seen input-signals for a few time-steps, so its generated output may be very inaccurate. Therefore, the model will be given a "warmup-period" of 50 time-steps where we don't use its accuracy in the loss-function, in hope of improving the accuracy for later time-steps, see [1]

Amazon Forecast DeepAR+ uses RMSE and WAPE metrics, RMSD Wikipedia article

I'm using this relation to compare the metrics for the Dilated Causal CNN and the Amazon DeepAR+ model: $RMSE = \sqrt{MSE}$

The re-sampled data will be used to create "future" period by shifting the target-data.

II. Analysis

Data Exploration

Weather data for the period 1936-2018 will be used for four cities in Bulgaria – Ruse, Sofia, Varna and Veliko Tarnovo. After struggling with the format of some of the local sources, I found that the National Centers for Environmental Information of the National Oceanic and Atmospheric Administration https://www.ncei.noaa.gov/ provide an access to a global source of climatic data. It was also used in [1], and one can pick almost any location in the world.

The dataset was generated using the old data request form here https://www7.ncdc.noaa.gov/CDO/cdo but not it forwards to a new online form, that lets one download station by station.

The dataset consists of 1 186 845 lines and 33 columns in fixed text format. The features / columns description is provided here: https://www.ncdc.noaa.gov/cdohtml//3505doc.txt

Most of these 33 features however have gaps and measurement data missing, replaced by ** symbols, especially for the older data. So, I decided to use only two columns: TEMP – temperature in deg F and SLP (surface level pressure) in hectopascals (hPa). This gives two time series per city/station, for the 4 selected cities.

Frequency of measurements however is not constant, that will need to be addressed in preprocessing – 2 times a day at 0600 and 1800 hours in the 1930-ies, up to to 48 times a day (every 30 minutes) the 2010s. A resampling is necessary and implemented in notebook 01, described later in III. Methodology.

The process is described in detail in the project notebook 01. The raw dataset was uploaded in ZIP to GitHub, for review and future use:
https://github.com/ogniandantchev/ML_engineer_nd009_capstone

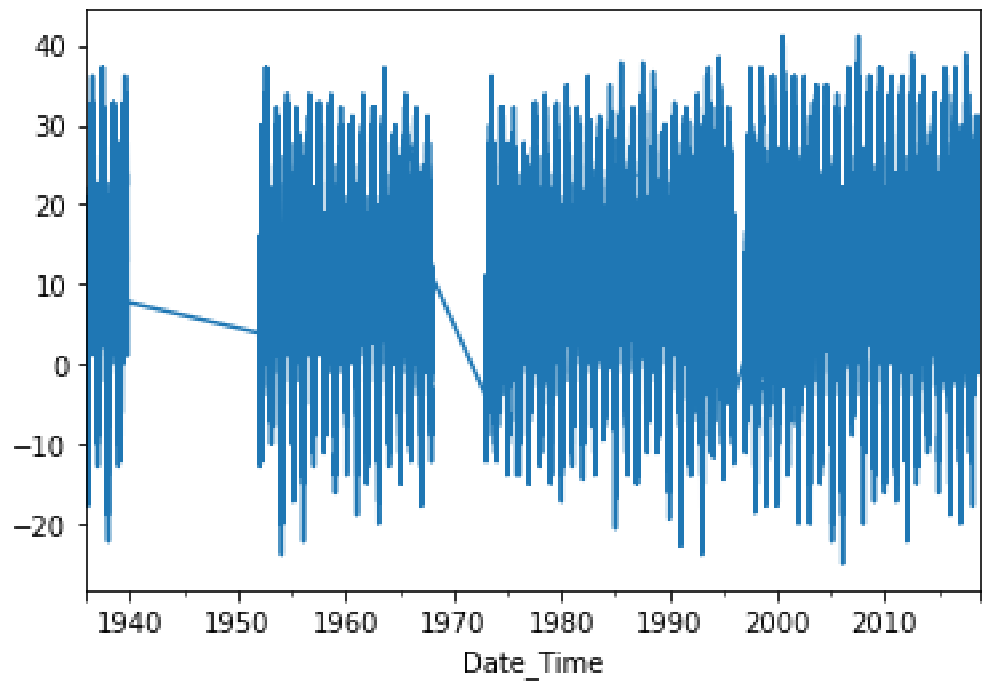
Bucharest
Craiova
Slatina
Slobozia
Călărași
Constanța
Năvodari
Eforie Nord
Costinești
Ruse
Русе
Pleven
Плевен
Veliko
Tarnovo
Велико
Търново
Vratsa
Враца
Varna
Варна
Sofia
София
Pernik
Перник
Bulgaria
Burgas
Бургас
Sozopol
Созопол
Pazardzhik
Пазарджик
Blagoevgrad
Благоевград
Plovdiv
Пловдив
Haskovo
Хасково
Bansko
Банско
Kardzali
Кърджали
Edirne
Serres
Σέρρες
Kavala
Καβάλα
Thessaloniki
Θεσσαλονίκη
Alexandroupoli
Αλεξανδρούπολη
Çorlu
Silivri
İstanbul

Bulgaria – Map with the 4 cities in the raw dataset

Since weather data is most diligently collected at airports, cities are encoded with their USAF codes. A function to convert these codes to names was prepared, as well as one that converts deg F to deg C – Data preprocessing below.

Exploratory Visualization

After the temperature conversion, one of the time series looks like this:



Sofia Temperature, 1936 to 2018

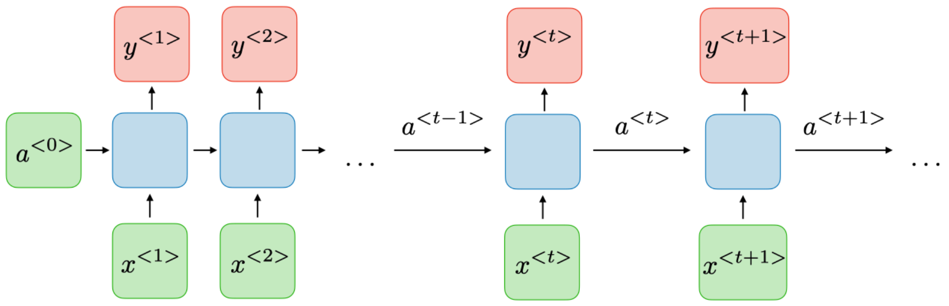
There are some gaps in the data, so I removed some period, and at the end I only use 1973 to 2018 – i.e. 45 years of data vs the original 82 years, but still more than enough. Removed 2 of the cities with bigger gaps, so only Sofia and Varna time series left in the clean dataset.

In addition, a minimum temperature of the equivalent of -71.67 deg C was found in the dataset, that is clearly an error, as there are no such temperatures around here in South Europe.

Algorithms and APIs

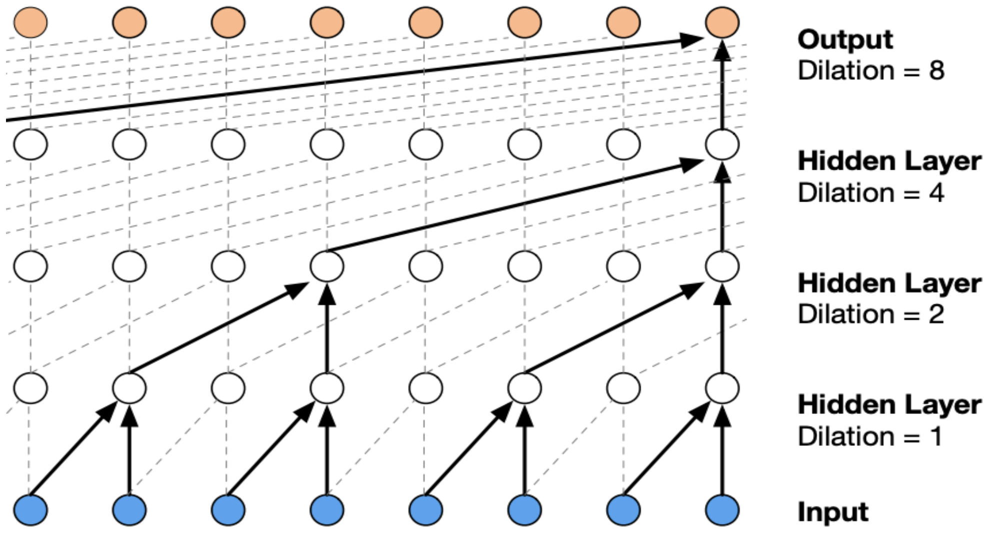
Recurrent neural networks (RNN) are a class of neural networks that is powerful for modeling sequence data such as time series or natural language. The 1st model will be based RNN with Gated Recurrent Unit (GRU). The 2nd one, I used in the past in [2], will be based on Dilated Causal CNN, often used for audio processing [6].

Here's a typical diagram of RNNs (from Stanford CS 230):



RNN

Here's a diagram of a Dilated Causal CNN, from the original WaveNet article [6]:



Dilated Causal CNN

III. Methodology

Data Preprocessing

As the raw dataset contains many gaps and the instances have different frequency if the time component, some preprocessing was needed. These steps were implemented in notebook 01, in the following sequence to acquire, clean and reshape the data:

- generate the raw data set at https://www.ncdc.noaa.gov/
- review the original dataset files
- resample data and helper functions
- remove data (where gaps are too big)
- add data (linearly interpolated from the neighboring values)
- save the clean, resampled data to a binary file.

One abnormal low temperature instance was deleted, and also 2 of the cities with big gaps in data were deleted. All data prior to 1973 was also removed.

After that, the series were re-sampled, so an hourly frequency can be used for the entire period (data was collected only 2 times a day in the 1930ies, then 3 times, then 6, and currently 48 times).

Pandas library .resample() and . interpolate() methods were used: first upsample to 30 min ('30T' parameter), then .interpolate(method='time'), then downsample to one hour frequency with .resample('60T') , as suggested in [1]. The result is a uniform time series, interpolated to have no gaps and with uniform frequency of 1 hour – very important for all NN models.

Since the data span is multiple years, in [1] they suggest 2 additional features to be engineered and added: day of the year (int64, values 1 to 366) and hour of the day (in64, 0 to 23). These 2 features will give the models a way to make sense of the seasonal and daily temperature variations. Here is the summary of the clean dataset:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 401784 entries, 1973-01-01 00:00:00 to 2018-11-01 23:00:00
Freq: 60T
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   (Varna, Temp, C)   401784 non-null  float64
 1   (Varna, SLP, hPa)  401784 non-null  float64
 2   (Sofia, Temp, C)   401784 non-null  float64
 3   (Sofia, SLP, hPa)  401784 non-null  float64
 4   (Various, Day)     401784 non-null  int64  
 5   (Various, Hour)    401784 non-null  int64  
dtypes: float64(4), int64(2)
memory usage: 21.5 MB
```

This was done in the project Jupyter notebook 01, and then the clean data was saved in Python Pickle standard binary file format.

Implementation

The Keras API and TensorFlow 2.5 were used to create the first 2 models, implemented in notebooks 02 and 03.

Gated Recurrent Unit (GRU) RNN Model Implementation

```
model = Sequential()
model.add(GRU(units=512,
              return_sequences=True,
              input_shape=(None, num_x_signals,)))
model.add(Dense(num_y_signals, activation="tanh"))
```

Loss function calculates MSE, but after ignoring some "warmup" part of the sequences:

```
warmup_steps = 50
```

Later on, these are visualized in gray in the Forecast plots.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
gru (GRU)                    (None, None, 512)         798720    
_________________________________________________________________
dense (Dense)                (None, None, 1)           513       
=================================================================
Total params: 799,233
Trainable params: 799,233
Non-trainable params: 0
___________________________________
```

Dilated Causal CNN Model Implementation

```
# Keras Sequential definition -- var model2

p='causal'

model2 = Sequential([
    Conv1D(filters=32, input_shape=( None, num_x_signals ),
           kernel_size=2, padding='causal', dilation_rate=1),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=2),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=4),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=8),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=16),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=32),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=64),
    Conv1D(filters=32, kernel_size=2, padding=p, dilation_rate=128),
    Dense(128, activation=tf.nn.relu),
    Dropout(.2),
    Dense(1, activation="tanh")
])

model2.compile(Adam(), loss='mean_absolute_error')
```

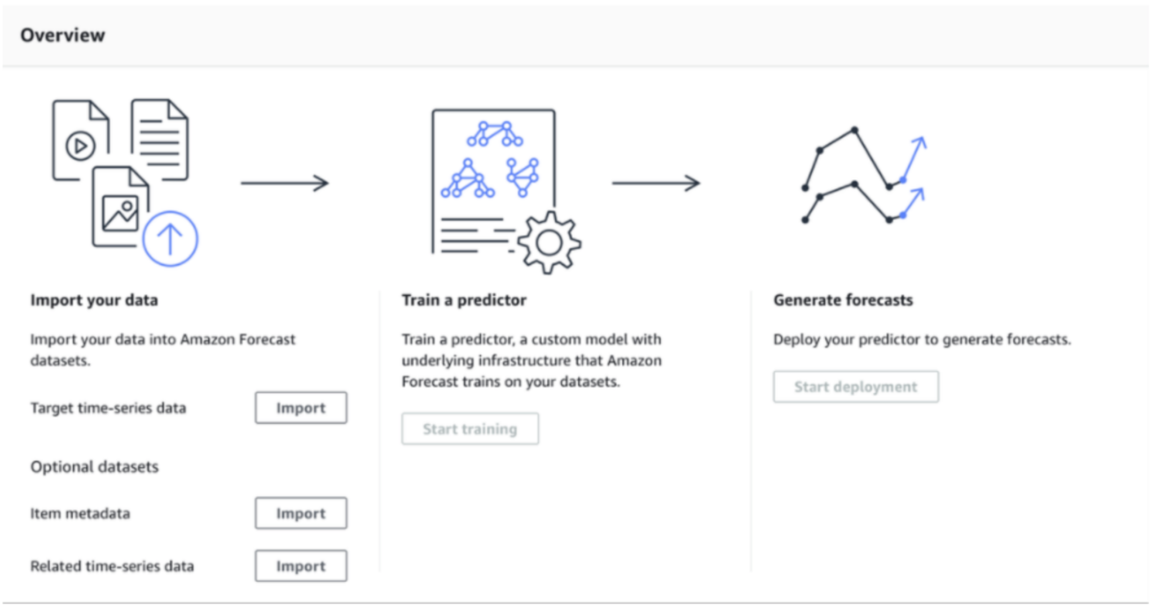
Using Standard MSE as loss here – `loss='mean_absolute_error'`.

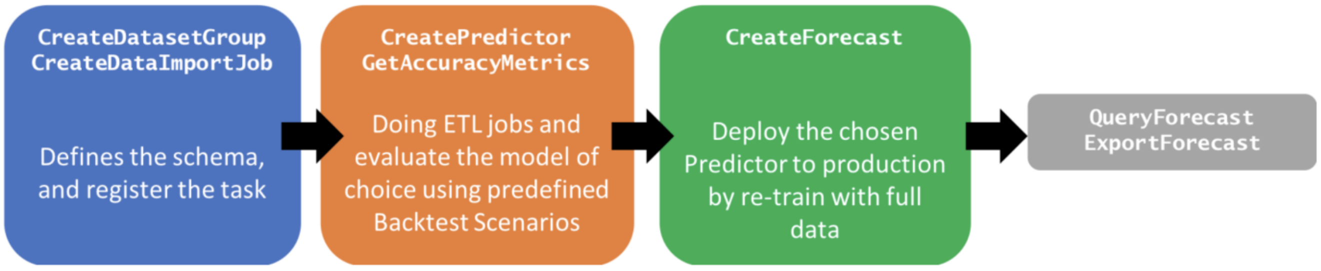
```
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_16 (Conv1D)           (None, None, 32)          416       
_________________________________________________________________
conv1d_17 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_18 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_19 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_20 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_21 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_22 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
conv1d_23 (Conv1D)           (None, None, 32)          2080      
_________________________________________________________________
dense_4 (Dense)              (None, None, 128)         4224      
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 128)         0         
_________________________________________________________________
dense_5 (Dense)              (None, None, 1)           129       
=================================================================
Total params: 19,329
Trainable params: 19,329
Non-trainable params: 0
```

Amazon Forecast model, based on DeepAR+ algorithm

DeepAR+ used in notebook 04, is an Amazon proprietary algorithm for forecasting scalar (one-dimensional) time series using RNNs. The process of using it is described in [3]:


Overview
Import your data
Import your data into Amazon Forecast datasets.
Target time-series data
Import
Optional datasets
Item metadata
Import
Related time-series data
Import
Train a predictor
Train a predictor, a custom model with underlying infrastructure that Amazon Forecast trains on your datasets.
Start training
Generate forecasts
Deploy your predictor to generate forecasts.
Start deployment


CreateDatasetGroup
CreateDataImportJob
Defines the schema, and register the task
CreatePredictor
GetAccuracyMetrics
Doing ETL jobs and evaluate the model of choice using predefined Backtest Scenarios
CreateForecast
Deploy the chosen Predictor to production by re-train with full data
QueryForecast
ExportForecast

Similarly, to the scheme above, most of the time and work is in preparing the dataset (one time series here) in the Amazon Forecast specific format, copy to the newly created S3 bucket, and import to AWS Forecast. Here's the import result at the end:

`forecast.describe_dataset_import_job(DatasetImportJobArn=ts_dataset_import_job_arn)`
(here I print only key selected data, please see notebook 04 for the complete output)

```
{'DatasetImportJobName': 'sof_temperature_forecast_1',
 'DatasetImportJobArn': 'arn:aws:forecast:eu-central-1:574930355514:dataset-import-
job/sof_temperature_forecast_1/sof_temperature_forecast_1',
 'DatasetArn': 'arn:aws:forecast:eu-central-1:574930355514:dataset/sof_temperature_forecast_1',
… 'DataSource': {'S3Config': {'Path': 's3://forecast-test-0/t1.csv',
   'RoleArn': 'arn:aws:iam::574930355514:role/ForecastNotebookRole-Basic'}},
…   'CountLong': 401784,
…  'target_value': {'Count': 401784,
…   'Min': '-25.0',
   'Max': '41.11',
   'Avg': 10.1715268852146,
   'Stddev': 9.428677566143127,
…   'Min': '1973-01-01T00:00:00Z',
   'Max': '2018-11-01T23:00:00Z',
… 'Status': 'ACTIVE',
…   'connection': 'keep-alive'},
…
```

Then creating the model, using the create_predictor method:

```
create_predictor_response = \
    forecast.create_predictor(PredictorName=predictor_name_deep_ar,
                              AlgorithmArn=algorithm_arn_deep_ar_plus,
                              ForecastHorizon=FORECAST_LENGTH,
                              PerformAutoML=False,
                              PerformHPO=False,
                              InputDataConfig= {"DatasetGroupArn": dataset_group_arn},
                              FeaturizationConfig= {"ForecastFrequency": DATASET_FREQUENCY}
                             )
```

```
forecast.describe_predictor(PredictorArn=predictor_arn_deep_ar)
```

This call to the AWS Amazon Forecast API gives description of the model created. Again, these are only key selected parameters, due to size, please see notebook 04 for the complete output:

```
{'PredictorArn': 'arn:aws:forecast:eu-central-1:574930355514:predictor/sof_temperature_forecast_1_deep_ar_plus',
 'PredictorName': 'sof_temperature_forecast_1_deep_ar_plus',
 'AlgorithmArn': 'arn:aws:forecast:::algorithm/Deep_AR_Plus',
 'ForecastHorizon': 72,
 'ForecastTypes': ['0.1', '0.5', '0.9'],…

'learning_rate': '1E-3',   'learning_rate_decay': '0.5',
  'likelihood': 'student-t',
… 'AutoMLAlgorithmArns': ['arn:aws:forecast:::algorithm/Deep_AR_Plus'],
 'Status': 'ACTIVE',
…
```

Creating forecast

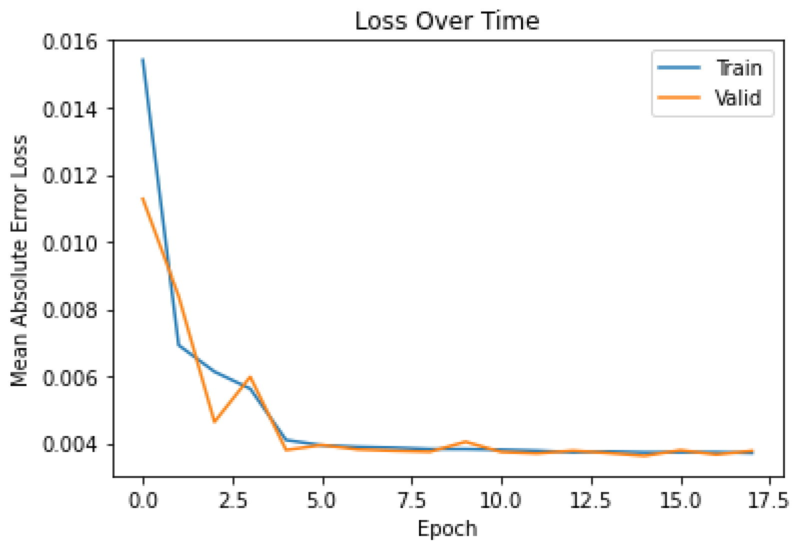
```
forecast.describe_forecast(ForecastArn=forecast_arn_deep_ar)

{'ForecastArn': 'arn:aws:forecast:eu-central-1:574930355514:forecast/sof_temperature_forecast_1_deep_ar_plus',
 'ForecastName': 'sof_temperature_forecast_1_deep_ar_plus',
 'ForecastTypes': ['0.1', '0.5', '0.9'],
 'PredictorArn': 'arn:aws:forecast:eu-central-1:574930355514:predictor/sof_temperature_forecast_1_deep_ar_plus',
 'DatasetGroupArn': 'arn:aws:forecast:eu-central-1:574930355514:dataset-group/sof_temperature_forecast_1',
 'Status': 'ACTIVE',
…
```

Refinement

The custom models in notebooks 02 and 03 were implemented as multivariate models, where related time series were used to give better prediction for the target series. The AWS proprietary model based on DeepAR+ algorithm, implemented in notebook 04 was used as-is, following the tutorial in [5], and as a result is the only model with one time series only. This is partially compensated by the fact this model is a much more complicated black box. In can be further improved by adding related time series, as described in the “advanced” section of [5], but this was left to be implemented as future improvement.

For the RNN model with GRU, in project notebook 02, an early stopping was implemented as callback, that resulted in a small reduction in training time (by 3 min), stopping at epoch 17/20:



IV. Results

Model Evaluation and Validation

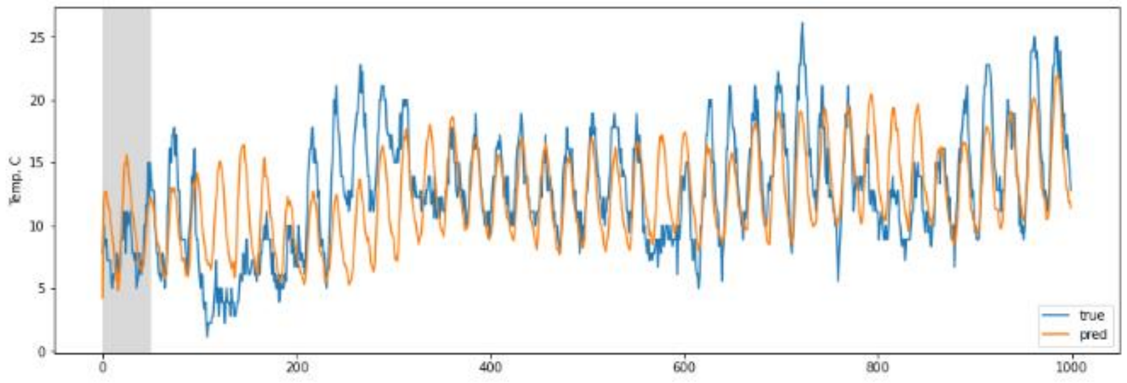
The training results, times, and the forecast plots are as follows:

Recurrent Neural Network with GRU:

```
Epoch 1/20
100/100 [==============================] - 61s 577ms/step - loss: 0.0154 - val_loss: 0.0113
…
Epoch 00017: val_loss did not improve from 0.00363
Epoch 18/20
100/100 [==============================] - 60s 599ms/step - loss: 0.0037 - val_loss: 0.0038

Epoch 00018: val_loss did not improve from 0.00363
Epoch 00018: early stopping
Wall time: 17min 54s
```

After inferencing and comparing the true signal (in blue) to the forecast (orange), this plot was generated with the function plot_comparison():



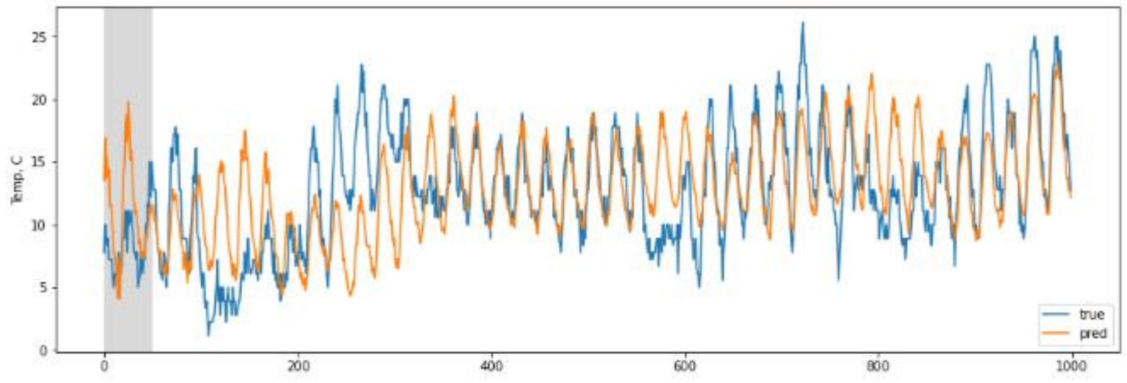
Gated Recurrent Unit (GRU) RNN Model forecast

Dilated Causal Convolutional Neural Network

This model was implemented using Keras and TensorFlow 2.5 and ran in the project notebook 03. Its training time was significantly shorter:

```
Epoch 1/10
100/100 [==============================] - 21s 184ms/step - loss: 0.0876 - val_loss: 0.0506 …
…Epoch 10/10
100/100 [==============================] - 18s 179ms/step - loss: 0.0520 - val_loss: 0.0479
Wall time: 3min 1s
```

And again, comparing true and predicted signal shows good agreement and very small MSE:

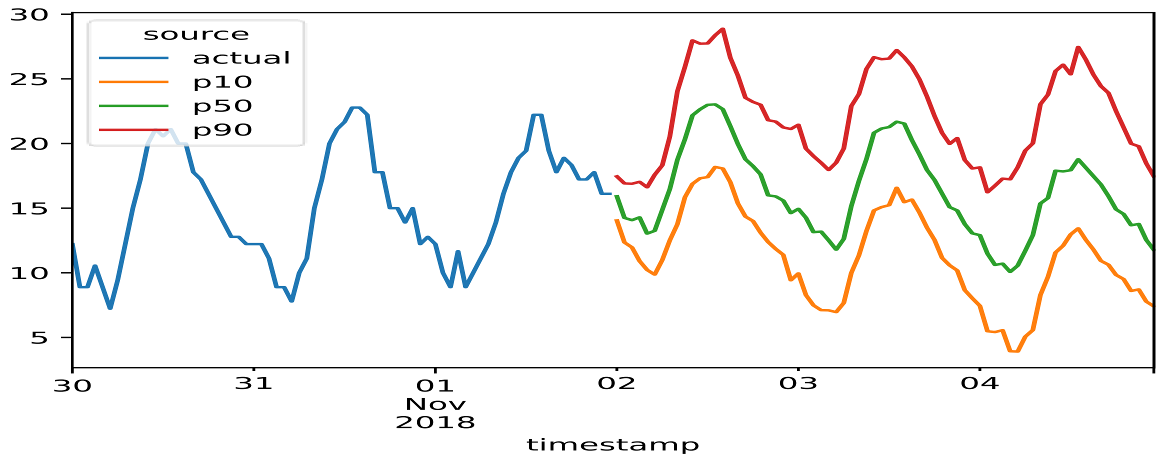


Dilated Causal CNN Model forecast

Amazon Forecast model, based on DeepAR+ algorithm

Here's the abbreviated result of a call to `forecast.get_accuracy_metrics` (see the full in notebook 04):

```
{'PredictorEvaluationResults': [{'AlgorithmArn': 'arn:aws:forecast:::algorithm/Deep_AR_Plus',
   'TestWindows': [{'EvaluationType': 'SUMMARY',
…     'Metrics': {'RMSE': 2.8056032983401455,
       'WeightedQuantileLosses': [{'Quantile': 0.9,
         'LossValue': 0.0640613829623758},
        {'Quantile': 0.5, 'LossValue': 0.13991594321957088},
        {'Quantile': 0.1, 'LossValue': 0.10678413658910633}],
       'ErrorMetrics': [{'ForecastType': 'mean',
         'WAPE': 0.14845030024660774,
         'RMSE': 2.8056032983401455}]}}]}],
…
```



Amazon Forecast DeepAR+ forecast.

Justification

The above plot was implemented in the Amazon Forecast tutorial and I recreated on the forecast my temperature dataset for the Sofia city only. DeepAR+ of Amazon Forecast gives probabilistic Monte Carlo type evaluations, calculating P10, P50 and P90 I.e., a statistical confidence level for an estimate, that I can't interpret and compare at the moment.

Comparing the main metric, MSE or RMSE, the Amazon model (notebook 04) has RMSE of 2.806, while the custom Dilated Causal CNN (notebook 03) has an MSE 0.0037 (or $RMSE = \sqrt{MSE} =$ 0.061). Of course, it is not fair to compare a model based on just one time series, to the multivariate custom model. But all models give fair results and can be used for various smaller time series for business outcomes (thee 45-year temperature time series are for illustration purposes only).

V. Conclusion

Reflection / Discussion

Two custom multivariate models for time series forecasting were implemented: RNN with Gated Recurrent Unit (GRU) and a Dilated Causal CNN, with Keras and TensorFlow. The 3rd based on Amazon Forecast DeepAR+ proprietary algorithm was implemented and deployed at AWS. An attempt was made to compare custom models to Amazon's own DeepAR+ model, but since they use different metrics, further work is required.

- The main goal was to learn how implement and deploy DeepAR+ model for time series forecast to AWS Amazon Forecast.
- The DeepAR+ model of Amazon Forecast (notebook 04) has RMSE of 2.806, while the custom Dilated Causal CNN (notebook 03) has an MSE 0.0037 (or $RMSE = \sqrt{MSE} = 0.061$). Of course, it is not fair to compare a model based on just one time series, to the multivariate custom model.
- the total cost of training and deploying the model at AWS Amazon Forecast was EUR 0.78 (USD 0.91) with the AWS free tier.

The state-of-the-art DeepAR+ model performs well, even though it was the only model using just one time series. The custom models are both multivariate, using temperature and pressure data for 2 cities + some engineered features to help make sense of time of day, and day of the year. All 3 models capture well the daily and seasonal variations in temperature.

Improvements / TODO

The implementation of 2 custom models and the Amazon proprietary DeepAR+ on the dataset made the project implementation too verbose. Here are some possible improvements I would implement in the future:

- implement a multivariate version of the predictor, based on the Amazon Forecast advanced examples here, and add the additional data from the original dataset – atmospheric pressure and temperature for other cities in the region
- at the moment metric for the model in 02 is MAE, in 03 metrics are MSE and MAPE, while Amazon Forecast uses RMSE and WAPE. need to have same for all 3 models.
- DeepAR+ of Amazon Forecast gives probabilistic Monte Carlo type evaluations, calculating P10, P50 and P90 – the plot at the end of Notebook 04. I.e., a statistical confidence level for an estimate. TODO: understand the concept and implement for the first two models.

VI. References

1. Magnus Erik Hvass Pedersen, TensorFlow-Tutorials / GitHub repo / Videos on YouTube
2. Ognian Dantchev, Multivariate Time Series Forecasting with Keras and TensorFlow, GitHub repo
3. Amazon Forecast resources, AWS website
4. Time Series Forecasting Principles with Amazon Forecast, Technical Guide
5. AWS Samples– Amazon Forecast Samples GitHub repo
6. Aaron van den Oord, Sander Dieleman et al., WaveNet: A Generative Model for Raw Audio