Time Series Forecasting with Neural Networks

We have built different types of NN models for time series forecasting covering forecasts for a single time step as well as for multiple steps. Our study includes linear, dense, CNN, RNN models, as well as a convolutional se2seq neural network modeled after Wavenet.

As our main objective was to learn more on how to build these networks using Tensoflow/Keras and how to transform the time series dataset to train and predict, we have decided to fuly develop these models from scratch without using the code made available to us during the exercise sessions. The code of all our models is available on github in the folder src_neural_network.

Our work is based on these two excellent publications:

- TensorFlow time series forecasting tutorial
- JEddy32's TimeSeries_Seq2Seq

Simple multi steps models

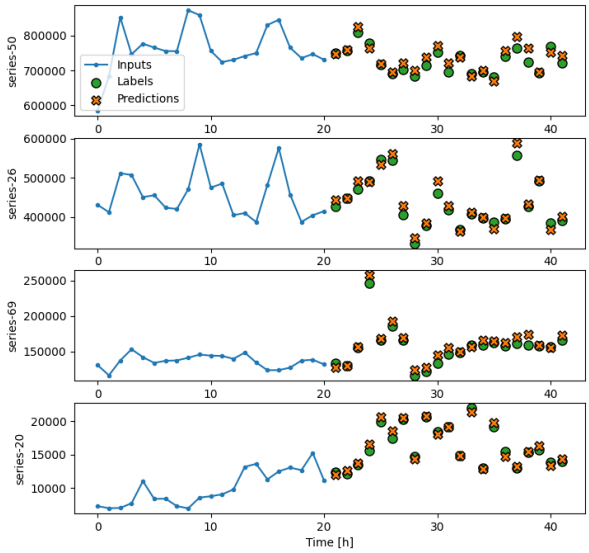
We have built several simple models to make multiple time step predictions. These models make "single shot predictions" where the entire period is predicted at once (i.e. 21 days). These models predict also all features (series) at once. The code is avalaible in the class MultiStepModels. As a example, here is the CNN model:

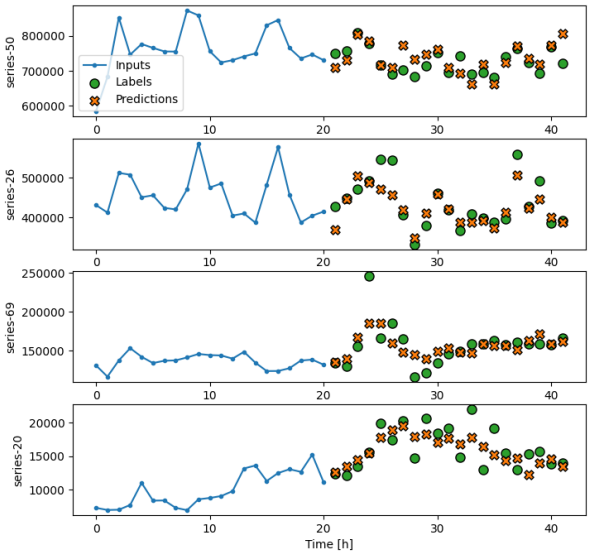
```
def model_dense(self):
    multi_dense_model = tf.keras.Sequential([
        # Take the last time step.
        # Shape [batch, time, features] => [batch, 1, features]
        tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
        # Shape => [batch, 1, dense_units]
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(1024, activation='relu'),
        # Shape => [batch, out_steps*features]
        tf.keras.layers.Dense(MS_OUT_STEPS*self.num_features,
                              kernel_initializer=tf.initializers.zeros()),
        # Shape => [batch, out_steps, features]
        tf.keras.layers.Reshape([MS_OUT_STEPS, self.num_features])
    ])

    history = compile_and_fit(multi_dense_model, self.multi_window)

def model_cnn(self):
    CONV_WIDTH = 3
    multi_conv_model = tf.keras.Sequential([
        # Shape [batch, time, features] => [batch, CONV_WIDTH, features]
        tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
        # Shape => [batch, 1, conv_units]
        tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
```

```
        tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
        # Shape => [batch, 1,  out_steps*features]
        tf.keras.layers.Dense(MS_OUT_STEPS*self.num_features,
                              kernel_initializer=tf.initializers.zeros()),
        # Shape => [batch, out_steps, features]
        tf.keras.layers.Reshape([MS_OUT_STEPS, self.num_features])
    ])

    history = compile_and_fit(multi_conv_model, self.multi_window)
```

The plots below show the predictions over the course of 21 days for these 2 for a few arbitrary series. The green dots show the target prediction value, the orange dots shows the actual prediction:

Dense Model	CNN Model





These models look to do a good job picking up on seasonality and trend, and handling the prediction horizon for many series. But the Kaggle scores are disapointing: around 17.5. So, we decided to focus on a more promising network: an autoencoder model using DeepMind's WaveNet concepts!

Forecasting with a convolutional sequence-to-sequence neural network modeled after WaveNet

The most promising network that we have modeled is a convolutional Seq2Seq neural network using DeepMind's WaveNet model architecture. This work is based on several readings of articles, and in particular J.Eddy's blog with its accompaying notebooks. Using the ideas and code developed by J. Eddy, we have trained a Wavenet-style network with a stack of 2 x 9 dilated causal convolution layers followed by 2 dense layers. Using 9 dilated convolution layers allows to capture over a year of history with a daily time series.

Here's the code defining the model:

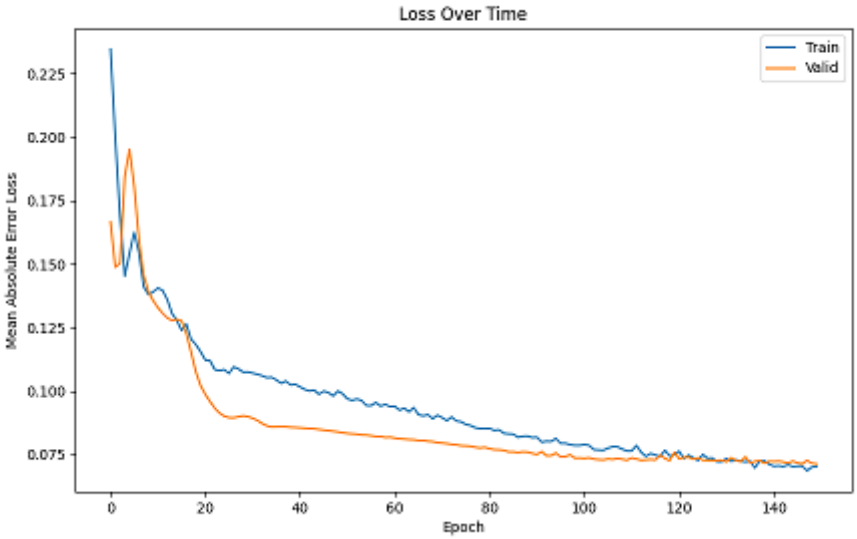
```
    def build_training_model(self):

            # convolutional operation parameters
            n_filters = S2S_CONVFULL_N_FILTERS # 32
            filter_width = S2S_CONVFULL_FILTER_WIDTH # 2
            dilation_rates = [2**i for i in range(S2S_CONVFULL_N_DILATIONS)] * 2 # 9
            n_dilation_layers = len(dilation_rates)
            n_dilation_nodes = 2**(S2S_CONVFULL_N_DILATIONS-1)

            # define an input history series and pass it through a stack of dilated causal con
            history_seq = Input(shape=(None, 1))
            x = history_seq

            skips = []
            for dilation_rate in dilation_rates:
                # preprocessing - equivalent to time-distributed dense
                x = Conv1D(n_dilation_layers, 1, padding='same', activation='relu')(x)

                # filter convolution
                x_f = Conv1D(filters=n_filters,
                             kernel_size=filter_width,
                             padding='causal',
                             dilation_rate=dilation_rate)(x)

                # gating convolution
                x_g = Conv1D(filters=n_filters,
                             kernel_size=filter_width,
                             padding='causal',
                             dilation_rate=dilation_rate)(x)

                # multiply filter and gating branches
                z = Multiply()([Activation('tanh')(x_f),
                                Activation('sigmoid')(x_g)])

                # postprocessing - equivalent to time-distributed dense
                z = Conv1D(n_dilation_layers, 1, padding='same', activation='relu')(z)

                # residual connection
                x = Add()([x, z])

                # collect skip connections
                skips.append(z)

            # add all skip connection outputs
            out = Activation('relu')(Add()(skips))

            # final time-distributed dense layers
            out = Conv1D(n_dilation_nodes, 1, padding='same')(out)
            out = Activation('relu')(out)
            out = Dropout(.2)(out)
            out = Conv1D(1, 1, padding='same')(out)

            pred_seq_train = Lambda(self.slice, arguments={'seq_length':HORIZON})(out)
```

```
    model = Model(history_seq, pred_seq_train)
    model.compile(Adam(), loss='mean_absolute_error')

    print(model.summary())

    return model
```

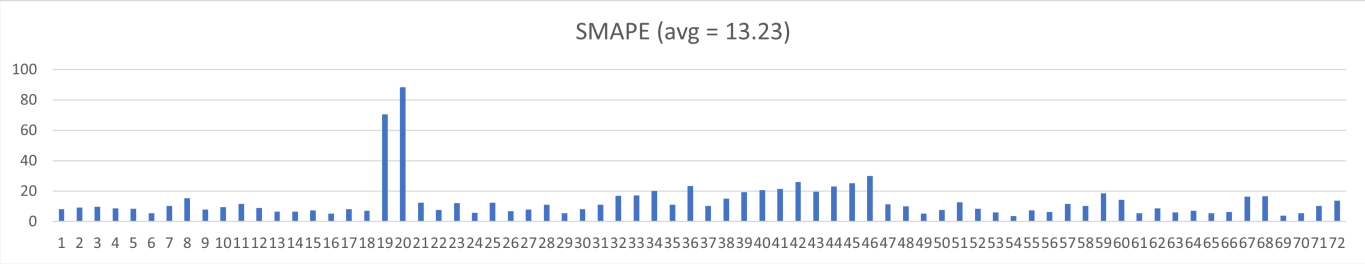
Before training this model, we have applied 2 transformations to the data:

1. Removed the outliners using a Hample filter with window_size=8, threshold=3
2. Applied a log1p transformation to smooth out the scale of traffic accross different series, and then centering the series around the mean of their training dataset.

This model trains quickly. The plot below shows the training convergence. We stopped training after 150 epochs.

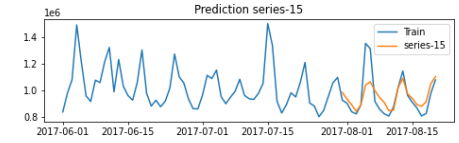
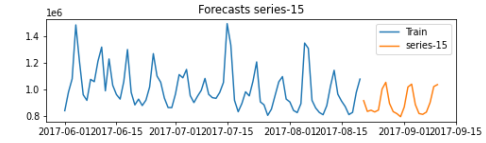
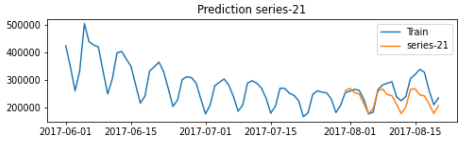
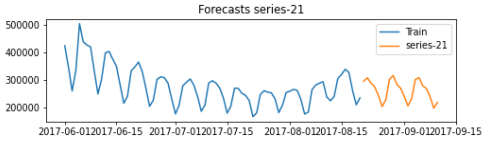
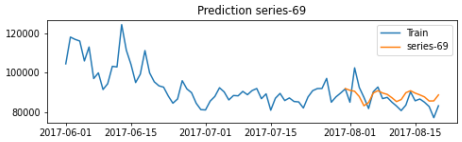
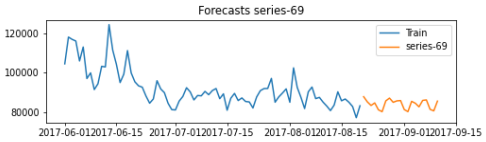
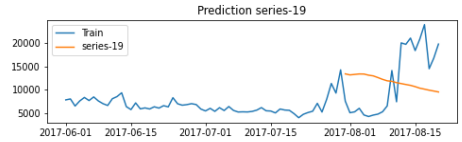
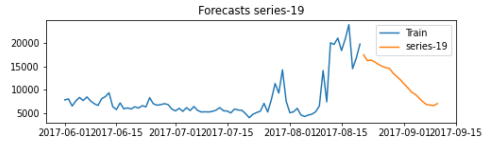
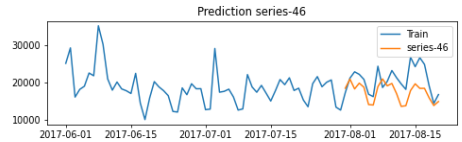
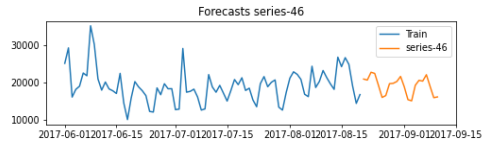


We have estimated the SMAPE value for each series with a 3-week prediction at the end of the training period (2017-07-31 to 2017-08-20):



The model does a good job picking up on seasonality and trend, and handling the prediction horizon for many series (SMAPE < 10). However, there is a significant number of series that are not properly modeled (e.g. series 19, 20, 40-46). The plots below give a few examples of predictions, as well as forecasting over the period 2017-08-21 to 2017-09-10.

SMAPE	Predictions	Forecasts

SMAPE	Predictions	Forecasts
series-15 smape=7.4		
series-21 smape=12.5		
series-69 smape=3.95		
series-19 smape=70.6		
series-46 smape=29.2		

It is worth mentioning that the main difficulties in developing this solution were in the steps before and after building and training the tensorflow model. Namely:

- Formatting the data for modeling: the time series must be first partitioned appropriately into encoding and decoding intervals; then additional transformation steps are required to extract the data into arrays that can be passed to the keras model's input layer. There is a nice explaination on how the data must be split and transformed in this blog. We took the code from Eddy's blog.
- Prepare for inference and forecast: many articles on autoencoders ignore this step. They explain how to build a model and train it but say nothing or very little on how to use the trained model for inference. Again, Eddy's blog was of a great help to define an inference architecture to feed the encoder and then have the decode generates a prediction for each time step. Something to note: we were not able to save the trained model and load it for inference in a seperate module. Tensorflow/Keras stopped with errors when loading the saved that we could not address. So, training and inference were done in sequences (which is not ideal of course).

Kaggle Competition

This model does not score well: 18.05. Not sure I understand why...

Summing Up

Well, the model's performance did not turn out the way we might have expected. The Kaggle score is not that good. The figures above indicate that our model can understand certain patterns but fail to capture the details of the variability (e.g. series-46). Abrut changes just before the forcast period are clearly not well taken into account (e.g. series-19). However, there are a number of reasons to consider it is possible to improve the results:

- We did not tune hyperparameters like dropout, loss, optimizer... It seems that the winner of the original Kaggle competition to predict Wikipedia web traffic did a lot of smart hyperparameter searches (in A. Nielsen's Pratical Time Series Analysis, O'Reilly).
- We did not try different encoder-decoder architectures. We could play with the number of dilation layers or the number of filters units.
- Maybe we did not explore the data enough and did not applied the best transformation before submitting the data to the model for training and inference.

Clearly, deep learning for time series forecasting is not a magic bullet.

References

- TensorFlow time series forecasting tutorial: https://www.tensorflow.org/tutorials/structured_data/time_series
- JEddy32 TimeSeries_Seq2Seq Github: https://github.com/JEddy92/TimeSeries_Seq2Seq
- Philippe Huet's defi3 github: https://github.com/tyxio/Web-Traffic-Time-Series-Forecasting