General Information

Type of Entry (*Academic, Practitioner, Researcher, Student*)	*Practitioner*
First Name	Slawek
Last Name	Smyl
Country	USA
Type of Affiliation (*University, Company-Organization, Individual*)	Company
Affiliation	Uber Technologies

Information about the method utilized

Name of Method	Exponential Smoothing - Recurrent Neural Network (ES-RNN) Hybrid Models
Type of Method (*Statistical, Machine Learning, Combination, Other*)	Combination
Short Description (up to 200 words)	Hybrid models, mixing hand-coded parts, like some Exponential Smoothing (ES) formulas, with a black-box Recurrent Neural Network (RNN) forecasting engine. The models do not constitute an ensemble of Exponential Smoothing (ES) and Neural Networks (NN) – instead, they are truly hybrid algorithms in which all parameters, like the initial ES seasonality and smoothing coefficients, are fitted concurrently with the NN weights by the same Gradient Descent method. Additionally, these are hierarchical models, in a sense that they use both global, applicable to all series, parameters (the NN weights) and per-time-series parameters (like seasonality and smoothing coefficients).

Neural Network for Time Series Forecasting

Generally, NNs do not have a great track record in time series forecasting. The series data needs to be preprocessed, and doing it properly is not straightforward. NN tend to have too many parameters (weights) to be fitted per each time series. Learning across many time series potentially solves the issue and opens possibilities of cross-learning, but it requires even more careful data preparation (especially normalization, often deseasonalization), and even with a very good preprocessing and good network architectures, the results tend to disappoint: It appears that such, trained across many time series, NNs, even Recurrent NNs, average their responses too much, in other words, the outputs are not enough time-series-specific.

The models described here try to solve this problem by being hierarchical – partly time series specific and partly global. Creating this kind of models has become possible thanks to recent appearance of Dynamic Computation Graph Neural Network systems (DGNNs), like Dynet and Pytorch, and the very recent "eager execution" mode in Tensorflow.

The algorithms

Deseasonalization

In the Competition, the time series were provided as vectors, without time stamps, so there was no obvious way to add calendar features, like day of week or month number. Additionally, there was little information provided on the source of the series, so there was no obvious way to group the series. For pure-time-series situations like this, NNs, also deep NNs, usually struggle to learn how to deal with the seasonality, and the standard approach is to deseasonalize the series using some statistical procedure like STL.

However, a deseasonalization algorithm, however robust and sophisticated, is designed to fulfill some statistical criteria that are likely to be only partly aligned with a goal of producing good features to NN with the overarching goal of good forecasting. It is better when deseasonalization is an integral part of the forecasting algorithm, as in case of e.g. Exponential Smoothing.

My seasonal models use algorithms inspired by Holt-Winters multiplicative seasonality, either single, or, in case of the hourly series, double seasonal (24 and 168). In the former case, the update formulas are

$$l_t = \alpha * y_t / s_t + (1- \alpha) * l_{t-1} \quad (1a)$$

$$s_{t+K} = \beta * y_t / l_t + (1-\beta) * s_t \quad (1b)$$

where l_t is *level,* y_t is value of the series, and s_t is the seasonality coefficient at the time t. K is the seasonality size, e.g. 12 for monthly series. α and β are smoothing coefficients, between 0 and 1, and $s_t>0$. For completeness, here is the forecasting formula:

$$\hat{\mathbf{y}}_{t+1..t+h} = \text{RNN}(\mathbf{x}_t) * l_t * \mathbf{s}_{t+1..t+h}, \quad (1c)$$

where $\mathbf{x}_t$ is the preprocessed (deseasonalized and normalized) input vector. The first K seasonal coefficients are also parameters of the model, fitted by the Gradient Descent. Multiplication in formula 1c is element-wise, bold letters denote vectors. The RNN forecasts whole horizon at once, e.g. 18 points for monthly series.

Feature extraction: windowing, normalization, and regressors

The feature extraction was rather simple and standard: we use rolling input and output windows of constant size, as in the Fig.1 below.

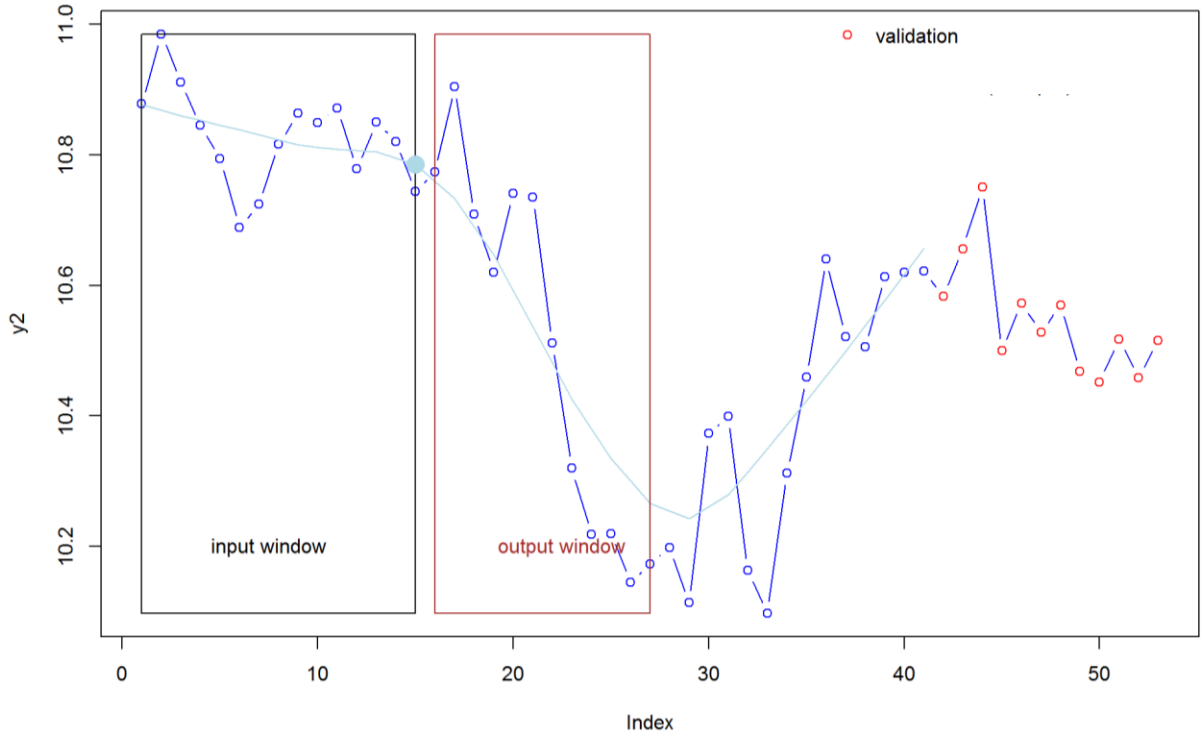


Fig.1 Feature extraction by rolling windows. The smooth curve depicts *level.*

The output size was always equal to the forecasting horizon, e.g. 13 for the weekly series, while input window size was established partly by a rule that it should cover at least full seasonality, e.g. has to be equal or larger than 4 in case of quarterly series, and partly by some experimentation. The reasoning behind the above rule was that the automatic deseasonalization may not be perfect, so the network should "see" at least seasonality-size number of past points at every step. For series without seasonality, e.g. yearly, the input window size was similar or a bit smaller than the forecasting horizon.

Generally, increasing input window size and extracting more sophisticated features, like strength of seasonality, variability etc., is a good idea, but I did not pursue it here for two reasons: the primary being that many series were relatively short, shorter or equal to the sum of the proposed input and output window sizes, therefore making impossible to forecast them. The secondary reason was hope the Recurrent NNs with their state and large amount of data will make this kind of advanced preprocessing not needed. However, as described above, in the Deseasonalization chapter, deseasonalization and normalization was conducted.

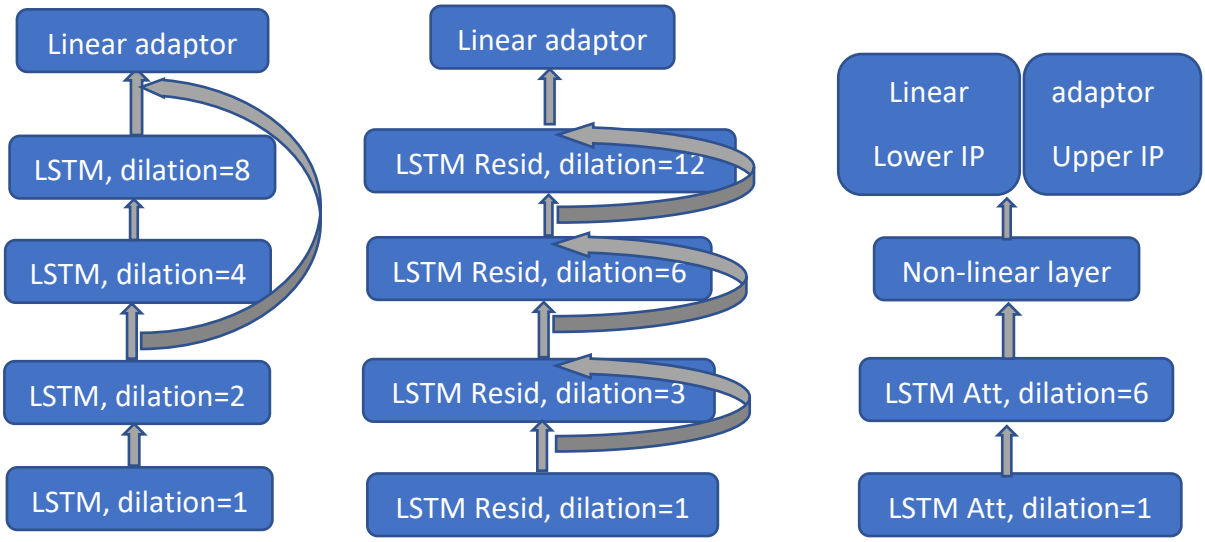
Additionally, series origin or type, e.g. Finance, Macro etc. was one-hot encoded as 6-long vector and appended to the time series-derived features.

Neural architectures

Several architectures were deployed, guided mostly by experimentation (backtesting results). At a high level they are all Dilated LSTM-based blocks, sometimes followed by a non-linear layer, and always followed by linear "adaptor" layer whose main job was to convert from a hidden-layer size to the output size (prediction horizon).

The LSTM blocks were composed of a number (1-2) of sub-blocks optionally connected with Resnet-style shortcuts. Each sub-block was a sequence of 1 to 4 layers, belonging to one of the three types of Dilated LSTMs: standard (Chang 2017), with Attention mechanism (Qin 2017), and a Residual version using a special type of shortcuts (Kim, El-Khamy and Lee 2018). Fig.2 shows three examples of configurations, for

a. Point forecast, quarterly series, which consists of two blocks, each of two dilated LSTMS, connected by a single "classical" shortcut around the second block
b. Point forecast, monthly series, which consists of a single block composed of 4 dilated LSTMs, with residual connections à la (Kim, El-Khamy and Lee 2018). Please note that the shortcut arrows point correctly into the inside of the Residual LSTM cell – this is a non-standard residual shortcut.
c. Prediction interval forecast, yearly series, which consists of a single block composed of two dilated LSTMs with Attention mechanism, followed by a dense non-linear layer (with tanh() activation), followed by a linear adaptor layer, of the size equal double of the output size, so we can forecasts both lower and upper quantiles at the same time.



(1,2)-(4,8) Std	(1-3-6-12) Residual à la Kim	(1,6),NL
Quarterly	Monthly	Yearly, Pred. Intervals

Fig.2 Example NN architectures.

Ensembling

There are several ensembling levels. Each member at any level except the lowest one, is an ensemble of members from a lower level.

1. Model level
   Ensembling forecasts from several independent runs (6-9) of a model. Each run produces one forecast per series.
2. Data subset level
   a. If practical in terms of computational requirements, use my Ensemble of Specialists algorithm. It creates a number of the forecasting models (e.g. 5) that specialize in some subset of the data, and then each series is forecasted by an ensemble, e.g. 3 or 4 of them. See my last year ISF talk https://forecasters.org/wp-content/uploads/gravity_forms/7-c6dd08fee7f0065037affb5b74fec20a/2017/07/smyl_slawek_ISF2017.pdf
   b. When it is impractical, for monthly and quarterly categories, split the data into two non-overlapping sets and fit and forecasts independently.
3. Stage of training level
   Average forecasts produced by a few (typically 5) most recent training epochs (per series).

Loss Functions

For point forecast I used Pinball loss operating on normalized and deseasonalized NN outputs and actuals. For reasons I do not quite understand, although using log() transformation probably played a role here, my models generally exhibited tendency for a positive bias. To counter it, the tau used was typically 0.49, but 0.45 for quarterly series, as they showed stronger tendency for a positive bias.

The Pinball loss function could have been also used for forecasting the prediction intervals, fitting separately models for 2.5% and 97.5%. Instead, mostly to reduce amount of work required, I encoded Mean Scaled Interval Score as the loss function. Of course, that also required changing the NN architecture to forecast at the same time both lower and upper quantiles. Again, the loss function operated on normalized and deseasonalized NN outputs and actuals.

Training details and Backtesting

As mentioned earlier some series were short, so a thorough backtesting by using rolling origin method was not feasible. I used last or penultimate horizon-size window of each series as the validation area.

On the other hand, some series were very long, having thousands of points. Stepping through say, last 300 years in order to forecast 6 years, seemed excessive. Also, it is not obvious that so old data is still relevant. There was also substantial computational cost involved with using all the data. So, I chopped the long series, e.g. used only last 20 years of monthly series and 40 years of quarterly series and 60 years of yearly series.

The forecasting system consists of 4 programs: two are using Ensemble of Specialists approach (see point 2a in the Ensembling chapter above) and two are using simple ensembling (point 2b there). Additionally, one member of each pair is for the point forecast and one for the prediction intervals. These are C++ programs relying on Dynet library, that can be compiled and run on Windows, Linux, and Mac. They can optionally write to a relational database (SQL Server or MySql). They are using CPU, not GPU, and are meant to be run in parallel. Please see comments inside the code.

References

Chang, Shiyu at al. 2017. "Dilated Recurrent Neural Networks." *NIPS.* Long Beach. https://papers.nips.cc/paper/6613-dilated-recurrent-neural-networks.pdf.

Kim, Jaeyoung, Mostafa El-Khamy, and Jungwon Lee. 2018. "Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition." *arxiv.org.* https://arxiv.org/abs/1701.03360.

Qin, Yao et al. 2017. "A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction." *arxiv.org.* https://arxiv.org/abs/1704.02971.