A Multi-Horizon Quantile Recurrent Forecaster
Paper review

Paper link: https://arxiv.org/pdf/1711.11053.pdf

Reviewers: Omri Kazelnik.

Abstract

Multi horizon forecasting is a modeling metadata able to forecast and predict multiple horizon terms attributes at once. For each horizon attribute, the Multi horizon contains a local prediction model.
The paper brings the power of general probabilistic time series regression, combined with quantile regression uncertainty additions and builds a full expressive and efficient solution for Direct Multi-Horizon Forecasting problem.
The paper findings may help in future designs of Encoder-Decoder style deep learning architectures and large scale complex multi horizons forecasting use cases.

Setup and Domain Considerations

The setup is Encoder-Decoder classic forecasting architecture.
The Decoder part is more complex compared to the multivariate features forecasting and DeepAR architectures in order to support multi horizon forecasting capabilities.
The authors targeted their framework to forecast many types of industrial streams, including but not limited to products demand forecasting and global energy consumption problems.

Encoder

The paper mainly discussed the Encoder part as vanilla LSTM, however there is part where a number of encoding extensions suggested to improve the performances.

NARX Encoder

The very basic reason for choosing LSTM architecture is to avoid the gradient vanishing problem RNN may suffer from. Because time serieses might encapsulate information with long horizon terms, LSTM sounds like reasonable architecture.
In addition, another applicable approach is NARX family. The family does not use the Markovian assumption on the relationship between the hidden states - namely h_t

computes based on h_t-1, but also sets a connection on the T past terms (also known as **skip connections**).
To gain the mentioned functionality, the Encoder hidden state outputs should be changed to summarize the requested T past states, meaning adding another dense linear layer to memorize these states.
In the button line, empirically it seems that the NARX-ish encode doesn’t bring any significant improvement over the vanilla architecture.

Paper Extensions

The main problem is in multi step long horizon time series might be composed of different streams with some calendar and other covariates.
The stream's joint distribution is hard to measure, and therefore evaluating the internals forecasting values and their respective uncertainties precisely requires estimating the risk error in decision making.
In addition, another major issue is to build a network with architecture that accounts for known future information, meaning adjustments in the time series covariate features.
The main benefits concluded from the paper are that prior proposed solutions tackle the issues aforementioned individually.
Compare to Encoder-Decoder architectures and particularly DeepAR, the authors provide a solution to partially multi horizon forecasting strategy (no samples path like DeepAR) which suppose to be much more efficient and directly generating accurate quantiles- where DeepAR just estimating the expectation error for conditionally Gussian process.

Why Quantiles

Multi horizon forecasting requires richer information for estimating the probability for time series value based on its history term events. To relieve the estimation difficulty, real valued time series solutions assume a residual error distribution (usually a Gussian), and if statisonarty is feasible this error reflects a Gussian process.
However, in many multi horizons cases exact (parametric) estimation is not feasible and therefore irrelevant, so adding uncertainty for the distribution parameters sounds like a reasonable solution.
In order to leverage the most wide uncertainty terms residual error, Quantile Regression brings robustness to the forecast by learns the conditional distribution P_q(y_t <= y_t|y_t-j) for each q quantile instead of estimating the exact parameters of the full distribution P(y_t <= y_t|y_t-j) for each time lag j.

Technical Aspects

The basic network architecture is a vanilla LSTM that encoding the history to hidden units h_t, while mimicking the Encoder-Decoder design of Seq2Seq.
In the Decoder part, DNN consists of different MLPs trying to forecast time slots by manipulating time contexts with future data.
Particularly, in MQ-RNN there are **two** levels of MLPs output dense flatten layers.

- The first MLP is a global context dense layer. It remembers for each quantile the encoder channel output, added to it the horizon context c_a which decodes common information from future K points and a series of horizon specific contexts c_(t+k) (where k is a continuous member of K).
- The second MLP is a local context dense layer. It basically combines for each forecast time slot T the two global contexts with the future horizon inputs and outputs K quantiles predictions.

Loss function

The loss function (e.g. comprehensive quantile loss) is an interpolation of the requested horizons, summing their weighted MAE over the forecast creation times (e.g. FCTs) with respect to Q different quantiles.

Training Procedure

The crucial difference between the paper over Seq2Seq is achieved by the training technique.
Unlike Seq2Seq traditional applications, where use **cutting-sequences** as training samples (cut stream in arbitrary point and split it to train and test), the paper suggested a new approach called **forking-sequences** for training MQ-RNN models.
The forking occurs for each arbitrary t (effectively, for each time stream length) the Encoder part propagates the stream/s in LSTM network, while the Decoder forecasts the future multi horizon streams and their respective global and local MLPs.
In a single train iteration (batch/none batch), MQ-RNN backpropagate through all the streams (e.g. FCTs) **together** saves time/complexity in the preprocessing train/test split phase and the need for data augmentation.
What makes the forking a robust and generalized solution is the fact that multi step horizon time series forecasting a single time slot resembles the general idea of ensembling weak learners and boosting them by adding an accurate **prior** on the auto-regression/correlation relationships between hidden states.

Training Details

The authors train their MQ-RNN on two datasets. The first is a dataset of weekly demand series on less than 60,000 products from years 2012-2016, where 2016 weeks (52) uses multi horizon forecasts (ranged from 1 to 52).
Furthermore, the authors added additional covariates features, mainly from three main families - static (for example calendar features), history (e.g. previous demands) and applicable future (e.g. promotions and discounts).
The second dataset deals with electricity consumptions and prices dataset (in daily grained intervals).
For the price prediction problem they used 168 hour duration, and for the electricity load problem 56 days.

Prediction Procedure

There is not much information about it in the paper, which might be interesting to investigate how the **forking-sequences** addition to classic Seq2Seq architecture affects the prediction process (managing the global contexts and local context changes).

Architecture Performs

The local MLP is the key point for being able to forecast more accurately by predicting the covariates templates. In vanilla Encoder-Decoder only horizon agnostic takes place, where it feeds recursively each cell by predecessors time slots. The advantage of the local MLP is losing the direct autoregression connection between successive observations (which might be high order polynomial or other function) and forecast based on Q quantiles derived from horizon specific context + encoding hidden states. Moreover, compared to DeepAR the uncertainty is more generalized, instead of relying on the joint likelihood of a single forecast horizon, while the local MLP estimating the Q quantiles joint distribution over K different horizons.

Metrics and Model Benefits

For metrics and model evaluations, the authors took *MQ_RNN* and compared it to different benchmarks. The basic benchmark model elected was *Seq2SeqC*, said as more general and efficient compared to DeepAR because it isn't restricted to an identical Encoder-Decoder (fact that I don't agree with, the identicality helps but isn't

architecture mandatory), and doesn’t need repeated sequential samples for forecasting horizons distributions.
The authors experiments results showed improvement over all benchmarks including *Seq2SeqC* with the best accuracy across all horizons.
The training loss of *MQ_RNN* converged faster without many oscillations (while *MQ_RNN_cut* fluctuates but indeed converged also).
The test error also showed the significant advantages of *MQ_RNN* different variations of the *Seq2SeqC* benchmark.

Prototype

Moving Parts

The main moving parts of the prototype creation will be probably Encoder and Decoder models.
To make things simpler, we can create the vanilla version of the Encoder with the basic LSTM network.
For the Decoder part, we will need to split the internal model to two MLP submodels - global MLP and local MLP.
A good short cut that should accelerate the prototyping is assuming there are no covariate features. Indeed it loses the advantages of generalizing the model, but doesn't hurt the general idea of improving the Decoder part over DeepAR and other benchmarks.

Training

The training of the Encoder is basically straightforward.
The training of the Decoder is effectively two networks composition training, where the loss function is sum quantiles error over all decoding horizons. An important note is the paper loss function takes into consideration only the **maximum** quantile error over all the batch samples, which is also an interesting discussion (why not taking the other differentiate (0,1) gradient function which reflects the whole batch error terms).
We need to optimize the loss over both two decoding MLPs and the Encoder LSTM.

Prediction

The prediction first invokes the Encoder part which returns the hidden states and outputs in time slot t.
The hidden state and output tensor will be passed unsqueezed to the global Decoder model and its outputs will be directed to the local Decoder to get a quantile/s prediction.

Employ Model

In our current product status we don’t see a “classic” point in the pipeline where we can add MQ-RNN naturally, but we might consider adding it as a new candidate to our multivariate forecasting internal models by replacing it with our univariate feature engineered model.

The replacement can save our feature engineering process and add covariate feature robustness.